Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

# Control de un agente inteligente mediante Redes Neuronales en el entorno del videojuego UT2004

PROYECTO DE FIN DE CARRERA

Autor: Sergio Moreno Ruiz

Director: Manuel González Bedia

Codirector: Francisco Serón Arbeloa

Ingeniería en Informática
Curso 2011-2012

Departamento de Informática e Ingeniería de Sistemas

Centro Politécnico Superior

Universidad de Zaragoza

Febrero de 2012

# Control de un agente inteligente mediante Redes Neuronales en el entorno de simulación UT2004

## RESUMEN

En este proyecto se pretende obtener agentes sintéticos (bots) para videojuegos de acción en primera persona, de forma que su comportamiento no sea definido directamente por el programador, sino que estos sean capaces de adquirirlo mediante aprendizaje automático. Para ello, se ha optado por una estrategia de aprendizaje basada en Redes Neuronales Recurrentes de Tiempo Continuo (CTRNN) (Beer, 1995a).

Las CTRNNs permiten al agente iniciar una acción independientemente de su situación inmediata y organizar su comportamiento anticipándose a eventos futuros (Beer, 1995b). Parte fundamental de este proyecto es que las CTRNNs sean capaces de aprender por sí mismas, para lo cual deben de ser capaces de adaptarse a un comportamiento dado mediante algoritmos genéticos y, si se requiere, de aprender y adaptarse a las circunstancias a lo largo del tiempo de ejecución del bot al que controlan.

El objetivo principal de este proyecto es el de estudiar y aprovechar las capacidades de las CTRNNs para obtener comportamientos para los bots de un videojuego de acción en primera persona que serían imposibles utilizando redes neuronales feed-forward (con comportamiento púramente reactivo). Para ello, se realizarán cuatro experimentos orientados a la obtención de cuatro bots controlados por CTRNNs:

1. En primer lugar se buscará obtener dos bots con diferentes comportamientos de navegación que requieran memoria a corto plazo: (a) un primer bot con comportamiento de navegación y evitación de obstáculos y (b) un segundo bot con capacidad de seguir la trayectoria de movimiento de un bot enemigo, incluso cuando lo pierde momentáneamente de vista al desaparecer éste tras un muro, para lo cual tendrá que poder "predecir" su reaparición.

2. En segundo lugar se buscará obtener un tercer bot con el que estudiar la capacidad de las CTRNN de aprender durante el tiempo de vida del bot sin variar ninguno de sus parámetros.

3. Por último, una vez estudiadas las propiedades de las CTRNNs para diferentes bots, se buscará obtener un cuarto bot cuyo comportamiento sea combinación de los obtenidos para el primer y el tercer bot.

El videojuego para el cual se programarán los bots es *Unreal Tournament 2004* (UT2004) el cual cuenta con la plataforma *Pogamut 3*, la cual permite programar el control de los bots en el lenguaje de programación *Java*. Al tratarse de un trabajo pionero, el objetivo previo a la realización de este proyecto consistirá en estudiar dicha plataforma (sus ventajas y limitaciones), así como realizar un manual adecuado para programadores (dado que *Pogamut*, al principio de este proyecto, no contaba con uno). Con ello se pretende asentar las bases para futuros proyectos en el área de IA en videojuegos utilizando *Pogamut*.

# Índice general

# Índice de figuras

# Índice de tablas

# Parte I

# Memoria

# Capítulo 1

# Introducción

## 1.1. Objetivo y alcance del proyecto

Actualmente, los desarrolladores empiezan a reconocer la necesidad de una mejora de la IA en los videojuegos, de forma que el comportamiento de los NPCs (Non-Player Characters) sea más impredecible y más parecido al de un jugador humano (Laird y van Lent, 2000). No obstante, todavía existe un fuerte contraste entre las técnicas de IA usadas en la industria del videojuego (entre las que destacan las máquinas de estados finitas, árboles de comportamiento y algoritmos de planificación) y otras utilizadas fundamentalmente en la investigación académica (más orientadas a métodos de aprendizaje automático, redes neuronales y algoritmos genéticos). Se cree que en el tránsito y adaptación de modelos académicos a entornos comerciales puede estar la clave de la nueva generación de los videojuegos del futuro (se puede encontrar un análisis sobre el estado actual de la Inteligencia Artificial aplicada a los videojuegos en el Anexo A).

En este proyecto se pretende obtener agentes sintéticos (bots) para videojuegos de acción en primera persona, de forma que su comportamiento no sea definido directamente por el programador, sino que estos sean capaces de adquirirlo mediante aprendizaje automático. Debe tenerse en cuenta que, además de ser necesaria una herramienta de aprendizaje adaptable y con capacidad de adaptarse a situaciones desconocidas, este tipo de videojuegos exigen una rápida reacción por parte del bot, por lo que será necesario que el procesamiento de la información se realice lo más rápido posible. Debido a ello, el aprendizaje basado en redes neuronales como controladores de los bots se muestra como una opción interesante a explorar.

Con excepción de juegos como *Creatures* o *Black&White*, la utilización de redes neuronales en videojuegos comerciales es prácticamente inexistente. No obstante, en el mundo académico ya se han dado los primeros pasos para obtener bots controlados por redes neuronales artificiales, como *NeuralBot* (Chapman, 1999), *NERO* (Stanley el al., 2005) y *DodgingBot* (Kadlec, 2008), entre otros. No obstante, todos ellos se caracterizan por utilizar redes neuronales feed-forward, las cuales soportan únicamente comportamientos reactivos. En contraste con este tipo de redes, en el mundo de la investigación es cada vez más frecuente la utilización de redes neuronales recurrentes y que utilizan el tiempo como uno de los múltiples soportes de la información que procesan.

Debido a su éxito en el control de agentes autónomos en trabajos anteriores de otros autores (Beer, 1996; Floreano y Mondada, 1994), en este proyecto se utilizará una estrategia de aprendizaje basada en Redes Neuronales Recurrentes de Tiempo Continuo (Continuous Time Recurrent Neural Networks, CTRNN) (Beer, 1995a). En dichas redes pueden existir ciclos en su estructura y son matemáticamente equivalentes a los distemas dinámicos. Las CTRNNs permiten al agente iniciar una acción independientemente de su situación inmediata y organizar su comportamiento anticipándose a eventos futuros (Beer, 1995b).

El objetivo principal de este proyecto es el de estudiar y aprovechar las capacidades de las CTRNNs para obtener comportamientos para los bots de un videojuego de acción en primera

persona que serían imposibles utilizando redes neuronales feed-forward. Debido a la necesidad de procesamiento de la información lo más rápidamente posible ya comentada, y dado que podría considerarse la posibilidad de combinar las redes obtenidas en un mismo bot, se buscará obtener las CTRNNs más pequeñas posibles que satisfagan los comportamientos deseados.

El videojuego para el cual se programarán los bots es *Unreal Tournament 2004* (UT2004). Dicha elección se debe a que es el videojuego utilizado en el concurso a nivel mundial denominado *2K BotPrize*, que consiste básicamente en una adaptación del test de Turing al dominio de los videojuegos. Además, UT2004 cuenta con la plataforma *Pogamut 3*, la cual permite programar el control de los bots en el lenguaje de programación *Java*. Al tratarse de un trabajo pionero, el objetivo previo a la realización de este proyecto consistirá en estudiar dicha plataforma (sus ventajas y limitaciones), así como realizar un manual adecuado para programadores (dado que *Pogamut*, al principio de este proyecto, no contaba con uno). Con ello se pretende asentar las bases para futuros proyectos en el área de IA en videojuegos utilizando *Pogamut*. Dicho manual puede consultarse en el Anexo D.

## 1.2. Contexto en el que se realiza el proyecto

A pesar de que, como se ha comentado anteriormente, en la industria del videojuego hay cierta reticencia a cambiar las técnicas de IA utilizadas tradicionalmente, en el mundo de la investigación académica ya se han dado los primeros pasos para la obtención de bots controlados por redes neuronales feed-forward. Algunos ejemplos de ello son los siguientes: *NeuralBot* (Chapman, 1999)), el primer bot en utilizar redes neuronales, diseñado para el videojuego Quake II; *NERO* (Stanley el al., 2005), un bot capaz de aprender en tiempo de ejecución cambiando los parámetros de la red neuronal, para lo que utiliza el algoritmo rNEAT (real-time NeuroEvolution of Augmenting Topologies) que permite cambiar además la topología de la red, y *DodgingBot* (Kadlec, 2008), primer bot programado utilizando Pogamut y que utiliza redes neuronales, siendo capaz de esquivar misiles en el videojuego UT2004.

Por otro lado, en los últimos años, entre la comunidad que estudia los agentes autónomos se ha manifestado un interés creciente en el uso de las CTRNNs (Beer, 1995a) para controlar el comportamiento de agentes (Beer, 1990) y en la evolución de las mismas como técnica de aprendizaje (Yamauchi y Beer, 1994; Floreano y Mondada, 1994). Estos trabajos, entre otros, han mostrado que la combinación de algoritmos genéticos y redes neuronales es una técnica muy interesante para desarrollar estructuras de control en agentes autónomos. En concreto, su uso de ha destacado en un área que se ha denominado Robótica Evolutiva (Harvey et al., 2005), la cual consiste básicamente en la aplicación de técnicas evolutivas a redes neuronales para obtener agentes autónomos en los que "emerjan" los comportamientos deseados. Aunque es cierto que inicialmente en los proyectos emprendidos se utilizaban agentes físicos, en los últimos años ha predominado la tendencia a implementar modelos adaptados al diseño de agentes sintéticos en entornos virtuales de simulación (Jakobi, 1998).

La obtención de agentes autónomos mediante entrenamiento de CTRNNs utilizando algoritmos genéticos, se inspira en los trabajos pioneros de Floreano y Mondada (1994) sobre comportamientos de navegación de robots móviles *Khepera* (Floreano y Mondada, 1994) en entornos no estructurados, y de Beer (1996), sobre la capacidad de las CTRNNs para diferentes comportamientos cognitivos sencillos (denominados "modelos mínimos de cognición"). También se han tenido en cuenta los estudios realizados por Izquierdo (2008), en los que demuestra que el comportamiento de una CTRNN está íntimamente ligado al modelado del bot y a su situación en el entorno, así como la capacidad de aprendizaje durante el tiempo de ejecución del agente por parte de CTRNNs sin plasticidad sináptica. Por último, para el análisis de las dinámicas de una CTRNN con su entorno, se han seguido las técnicas estudiadas y demostradas por Beer (1995a).

## 1.3. Metodología: CTRNNs que aprenden

Parte fundamental de este proyecto es que las CTRNNs sean capaces de aprender por sí mismas, para lo cual deben de ser capaces de adaptarse a un comportamiento dado y, si se requiere, de aprender y adaptarse a las circunstancias a lo largo del tiempo de ejecución del bot al que controlan.

En la naturaleza, la evolución por un lado, y el aprendizaje durante el tiempo de vida por otro, son las dos formas más importantes de adaptación biológica. Como es evidente, éstas operan en diferentes escalas de tiempo: mientas que la evolución permite a las poblaciones de individuos adaptarse lentamente a las necesidades y cambios del entorno, a su vez cada individuo necesita adaptarse a los cambios que ocurren durante su tiempo de vida.

**a) El aprendizaje evolutivo como aprendizaje del comportamiento para el bot**

A la hora de diseñar una red neuronal, podemos definir dicha red en función del número de nodos que la componen y las conexiones entre los mismos. No obstante, resulta inviable determinar los parámetros de la red (tales como el valor de los pesos las conexiones, de las bias, etc.) que definen un determinado comportamiento según crece la dificultad del problema y, con ello, el número de variables que determinan el sistema. Debido a ello, se utiliza el aprendizaje mediante técnicas como las Estrategias Evolutivas y Algoritmos Genéticos. Éstas permiten solucionar un problema de la siguiente manera:

1. Se crea una población de redes neuronales en la que a los parámetros de cada red se asignan valores aleatorios.
2. Se evalúan las posibles soluciones y se combinan las mejores para crear una nueva generación.
3. Para cada generación se repite el proceso de selección de las mejores soluciones y se repite el proceso de mezcla durante el número de generaciones deseado o hasta obtener un individuo que se comporte satisfactoriamente según los criterios del programador.

Este planteamiento se puede interpretar como un sistema de búsqueda de soluciones que intenta utilizar las mismas técnicas que la naturaleza ha encontrado ante problemas semejantes. Debe tenerse en cuenta que el comportamiento que se obtiene no depende únicamente de la red, sino que es producto de la interacción entre las dinámicas internas del agente, su cuerpo y su entorno.

En este proyecto, se utilizará el aprendizaje evolutivo para obtener las CTRNNs con los comportamientos deseados para el control de los bots. Concretamente, el algoritmo genético utilizado el de Evolución Diferencial (Prize, 1999), el cual se explica en profundidad en el Anexo C.

**b) El aprendizaje durante el tiempo de vida del agente como comportamiento del bot**

Una vez sintetizada la CTRNN para un comportamiento dado mediante aprendizaje evolutivo, puede ser deseable que la red neuronal obtenida siga siendo capaz de aprender durante el tiempo de vida del bot. Tradicionalmente, el aprendizaje ha sido asociado a la modificación de los parámetros de una red neuronal, especialmente los que involucran cambios en las conexiones sinápticas o los pesos de las conexiones de la red. No obstante, este tipo de asunciones no son necesarias y es posible sintetizar CTRNNs sin plasticidad sináptica (es decir, cuyos parámetros permanecen invariables) con capacidad de aprendizaje durante el tiempo de vida del bot (Izquierdo 2008). En estas circunstancias, se considera el aprendizaje como el comportamiento para el cual ha sido evolucionada la CTRNN. Este comportamiento de aprendizaje por parte de las CTRNNs se estudia en el capítulo 4.

## 1.4. Trabajo a realizar

Para la realización de este proyecto, se realizarán cuatro experimentos orientados a la obtención de cuatro bots controlados por CTRNNs, con el objetivo de estudiar las capacidades de este tipo

de redes como controladores de agentes sintéticos en el videojuego UT2004:

1. En primer lugar se buscará obtener dos bots con diferentes comportamientos de navegación que requieran memoria a corto plazo, para lo que deberán hacer uso de las recurrencias entre los nodos de la red.

   *a*) Para el primer bot se buscará comportamiento de navegación y evitación de obstáculos en el entorno no estructurado de UT2004 (se entiende como entorno no estructurado a un entorno en el que no es viable que un agente pueda disponer de un mapa por lo complejo o lo cambiante del mismo (Arkin, 1998)).

   *b*) En cuanto al segundo bot, se desea que sea capaz de seguir la trayectoria de movimiento de un bot enemigo, incluso cuando lo pierde momentáneamente de vista al desaparecer éste tras un muro, para lo cual tendrá que poder "predecir" su reaparición.

2. En segundo lugar se buscará obtener un tercer bot con el que estudiar la capacidad de las CTRNN sin plasticidad sináptica de aprender durante el tiempo de vida del bot sin variar ninguno de sus parámetros.

3. Por último, una vez estudiadas las propiedades de las CTRNNs para diferentes bots, se buscará obtener un cuarto bot cuyo comportamiento sea combinación de los obtenidos para el primer y el tercer bot. Para ello, la CTRNN resultante deberá ser capaz de alternar entre entre el comportamiento de uno y otro según la situación en la que se encuentre.

## 1.5. Herramientas utilizadas

Para la implementación de los bots se ha utilizado el videojuego *Unreal Tournament 2004* junto con la ampliación *GameBots2004* (que permite ejecutar bots en el videojuego) y la plataforma *Pogamut 3* (http://diana.ms.mff.cuni.cz/main/tiki-index.php) (que permite programar al agente virtual en el lenguaje de programación *Java* (utiliza *JDK 6*) y conectarlo y recibir información del videojuego mediante un plugin para *Netbeans 6.9.1*), cuya arquitectura puede verse en la Figura 1.1. El programa *UnrealED*, incluido en la instalación del videojuego, fue utilizado para el diseño de mapas adecuados para cada experimento.

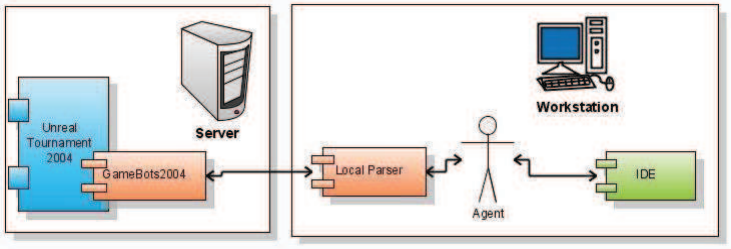



Figura 1.1: Arquitectura de Pogamut, donde "IDE" es NetBeans 6.9.1 y "Local Parser" se trata de un middleware entre GameBots2004 y el cliente, cuyo propósito es simplificar el envío y recepción de mensajes de GameBots2004 y minimizar la utilización del ancho de banda transmitiendo únicamente la información que ha cambiado.

Se contempló la posibilidad de adaptar la plataforma *JGAP* (Java Genetic Algorithm Plataform) (http://jgap.sourceforge.net/) para la evolución de las CTRNNs en lenguaje de programación *Java*. No obstante, su utilización tuvo que ser desechada, ya que JGAP estaba diseñada únicamente para evaluaciones rápidas, es decir, de menos de un segundo, ya que aumentaba enormemente el tamaño de cada población de individuos del algoritmo evolutivo, algo inviable para evaluaciones

que pueden durar hasta un minuto en este caso. Se tomó, por tanto, la decisión de diseñar módulos propios para la evolución de las CTRNNs.

Por último, para la obtención de las gráficas necesarias para el análisis de las dinámicas de las CTRNNs, se ha utilizado el módulo *Dynamica* (http://mypage.iu.edu/~rdbeer/) para el programa *Mathematica*, desarrollado por Randall Beer (1995a).

## 1.6. Estructura del documento

La estructura de esta memoria está dividida en cinco capítulos, además de este capítulo introductorio. En el capítulo 2 se expondrán y justificarán las técnicas y herramientas utilizadas para la realización de este proyecto, tales como el entorno de simulación UT2004, las CTRNNs y el algoritmo genético utilizado, así como sus ventajas y limitaciones. Los capítulos 3, 4 y 5 estarán dedicados a los experimentos descritos en los puntos 1, 2 y 3 de la sección 1.4 respectivamente. Por último, en el capítulo 6 se recogen las conclusiones extraídas a lo largo de los experimentos que componen el proyecto y se proponen las pautas a seguir para trabajos futuros.

## 1.7. Planificación

Durante los 16 meses de duración del proyecto, se han realizado tareas de documentación, redacción del manual de *Pogamut*, implementación de los módulos para los experimentos en *Pogamut*, ejecución y análisis de los experimentos, y la redacción de una presentación sobre IA en los videojuegos para su exposición en las charlas de la asignatura de Informática Gráfica. El diagrama de Gantt correspondiente a la realización del proyecto se muestra a continuación en la figura 1.2.

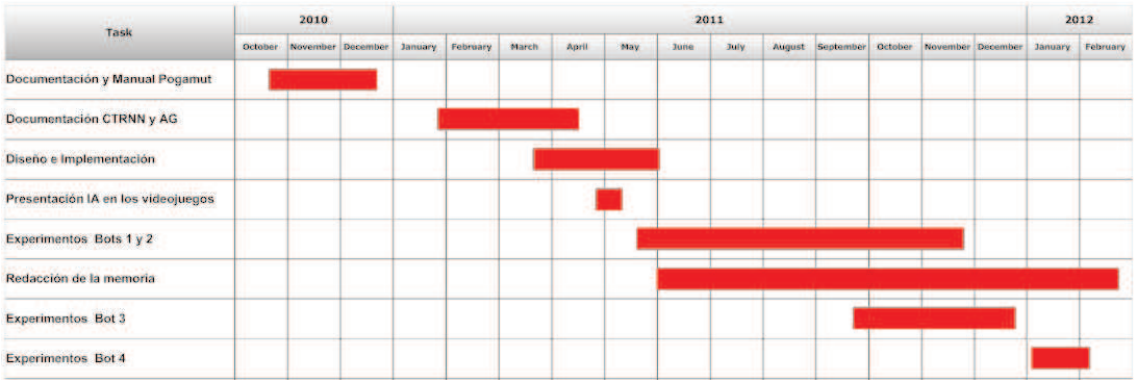



Figura 1.2: Diagrama de Gantt de las actividades realizadas.

# Capítulo 2

# Técnicas para la evolución de redes neuronales dinámicas en el entorno de simulación UT2004

En este capítulo se analizan en profundidad las técnicas y herramientas utilizadas para la realización de los experimentos de este proyecto. En la sección 2.1 se analiza el entorno de simulación del videojuego UT2004, así como la plataforma *Pogamut*. En la sección 2.2 se exponen las Redes Neuronales Recurrentes de Tiempo Continuo como controladores de bots. En la sección 2.3 se estudia el algoritmo de Evolución Diferencial (Prize, 1999). Por último, en la sección 2.4 se muestra el diseño a seguir para los experimentos de los siguientes capítulos.

## 2.1. El entorno de simulación UT2004 y Pogamut 3

El videojuego UT2004 proporciona todas las herramientas necesarias para la creación de los experimentos. La ejecución de bots en UT2004 se realiza por medio del mod *GameBots 2004* (GB or *GameBots*) y el editor *Unreal Editor* permite crear mapas personalizados para la simulación de los experimentos. La infraestructura de los bots ha sido programada en *Java* y conectada a UT2004 a través de la plataforma *Pogamut 3*, gracias a la cual se simplifica el desarrollo del bot y se reduce el tiempo necesario para depurar su comportamiento. Pese a las ventajas que esta plataforma ofrece a la hora de programar nuestros bots para UT2004, la utilización de este entorno de simulación supone una serie de limitaciones que deberán tenerse en cuenta a la hora de diseñar nuestros experimentos:

- UT2004 es un entorno en tiempo real, lo que supone que, a pesar de que el flujo del tiempo puede ajustarse, no existe una opción "correr a la máxima velocidad posible", lo cual sería muy útil para reducir el tiempo necesario para las evaluaciones del algoritmo genético (Kadleck, 2008).
- Un incremento en la velocidad estándar del juego provocaría fallos en el comportamiento de los bots, ya que no se terminarían de ejecutar todas las instrucciones que determinan su comportamiento, y se producirían por tanto resultados erróneos (Kadleck, 2008).
- Los tiempos de ciclo en UT2004 son irregulares. Pese a que es posible configurar manualmente cada cuánto tiempo *GameBots* debe ejecutar sus comandos de acción (el cual viene predefinido a 0.25 segundos, es decir, 4 acciones por segundo), la mala gestión por parte de *Pogamut* provoca alternancias entre dos valores (siendo el más habitual un tiempo de ciclo

irregular entre 0.4 y 0.5 segundos, y el menos habitual es un tiempo de ciclo también irregular entre 0.2 y 0.25 segundos).

- Debido a la mala gestión de los recursos por parte de *Pogamut*, en ocasiones no terminan de ejecutarse todas las instrucciones y cálculos necesarios para obtener las salidas de cada una de las redes neuronales, por lo que el comportamiento de dichos bots es erróneo e incluso se llegan a producir errores irrecuperables en su ejecución.

- Pogamut permite ejecutar varios bots en paralelo en la misma computadora mediante una sola instrucción. No obstante, una vez ejecutados dichos bots, deberemos esperar a que termine su ejecución para poder lanzar el siguiente "paquete" de bots. Esto impide la ejecución en paralelo de varios "paquetes" de bots, tanto en la misma computadora como en computadoras diferentes.

*Pogamut 3* se encuentra todavía en versión beta y al comienzo de este proyecto no contaba con un manual para programadores, por lo que la primera parte de este proyecto consistió en la elaboración del mismo. Dicho manual, el cual puede consultarse en el Anexo D, está basado en programas de ejemplo y el *Javadoc* correspondiente a la plataforma, y en él se explican todas las herramientas y funcionalidades que ofrece *Pogamut* y cómo utilizarlas.

## 2.2. Redes Neuronales Recurrentes de Tiempo Continuo

En contraste con las redes neuronales feed-forward, las cuales soportan únicamente comportamientos reactivos, en las Redes Neuronales Recurrentes de Tiempo Continuo (CTRNN) (Beer, 1995a) pueden existir ciclos en su estructura y la activación de sus neuronas es asíncrona y multi-escalada en el tiempo. Este tipo de redes neuronales también facilita describir el agente como un sistema dinámico acoplado al entorno en el que está ubicado, ya que está demostrado que son el modelo más simple de red neuronal dinámica continua no lineal (Funahashi y Nakamura, 1993). Además, la interpretación neurobiológica de las CTRNN ha sido demostrada y puede consultarse en (Beer, 1995a).

#### Descripción matemática de una CTRNN

Las CTRNN están formadas por neuronas cuyo comportamiento se describe en la ecuación

$$\dot{y}_i = \frac{1}{\tau_i} * \left( -y_i + \sum_{j=1}^{N} w_{ji} * \sigma(y_j + \theta_j) + I_i \right) \qquad i = 1, 2, \ldots, N \tag{2.1}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

donde $y_i$ es el estado de la neurona, $w_{ji}$ es el peso de la conexión entre las neuronas $i$ y $j$, $\theta$ es el término bias, $I$ representa una entrada externa y $\tau$ hace que cada una de las neuronas dependa del tiempo, ya que para diferentes valores la caída del nivel de activación de la neurona es más rápida o lenta. En la fórmula 2.1 la velocidad de actualización de la red neuronal debe ser notablemente mayor (el intervalo entre dos actualizaciones será menor) que el valor de $\tau$ para no obtener comportamientos no deseados.

En el Anexo B se puede encontrar de manera más detallada la descripción matemática de una CTRNN.

**Valores de activación y de salida de la neurona de una CTRNN**

Para poder entender cómo deben interpretarse la activación y la salida de una CTRNN, se va a utilizar una CTRNN formada por una única neurona autoconectada como la de la figura 2.1-A.

El valor de salida $o$ de una neurona será un valor real entre 0 y 1 obtenido al aplicar la función sigmoidea (ecuación 2.2) a la suma del estado actual $y$ de la neurona con su valor bias $\theta$, tal y como puede verse en la figura 2.1-B.

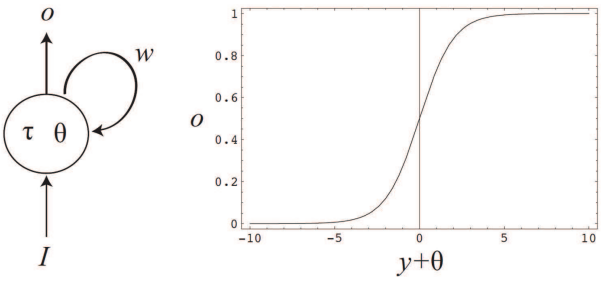



Figura 2.1: Valor de salida en una CTRNN. **Izquierda:** Un nodo autoconectado. **Derecha:** Función sigmoidea aplicada para calcular la salida de una neurona.

En cuanto al valor de activación, a diferencia de una red neuronal feed-forward, la cual realiza un mapeo directo entre entrada y salida de la red, el comportamiento de una CTRNN corresponde al de un sistema dinámico (Beer, 1995a), por lo que el valor de activación de la neurona convergerá a un punto de equilibrio. Para el análisis de las dinámicas del sistema formado por el agente controlado por la CTRNN y el entorno, se analizarán sus diagramas de bifurcación, los cuales muestran todos los puntos de equilibrio para la activación de las neuronas de la red. Para saber más acerca de la activación de las neuronas y el análisis de las dinámicas de un agente controlado por una CTRNN con su entorno se puede consultar el anexo B.3.

**Modelos de interconexión sináptica y simetría bilateral**

Para la realización de los experimentos se utilizarán CTRNNs en las que todas las neuronas estén totalmente interconectadas (conexión de cada neurona con todas las demás en ambas direcciones), autoconectadas (conexión recurrente de la neurona a sí misma) y conectadas a su vez a todos los sensores (todas las neuronas reciben las entradas de los sensores del bot). Además, se buscará obtener redes neuronales lo más pequeñas posibles que proporcionen un comportamiento satisfactorio.

Otro tipo de redes neuronales muy utilizadas en trabajos de otros autores (Floreano y Mondada, 1994; Beer,1996) son las llamadas redes de Elman. Éstas se explican en el Anexo B, en caso de que puedan ser útiles para futuros proyectos en los que se trabaje con CTRNNs.

En caso de ser posible, se utilizará la técnica de simetría bilateral, gracias a la cual se puede mantener reducido el número de parámetros a evolucionar. El motivo por el cual podemos aplicar esta técnica, es que algunos de los comportamientos de navegación en este trabajo son intrínsecamente simétricos, es decir, en ellos se espera que un patrón sensorial percibido en el lado izquierdo del bot produzca un activación simétricamente idéntica a la misma entrada sensorial percibida en el lado derecho.

## 2.3. Evolución Diferencial

En un problema que admita una solución basada en algoritmos genéticos se deben escoger aquellas técnicas evolutivas que mejor se adapten al tipo de controlador utilizado. Recientemente, se han realizado estudios que prueban que la técnica de Evolución Diferencial (Prize, 1999) puede

ser el candidato más óptimo cuando se pretende modelar sistemas dinámicos no lineales utilizando CTRNNs (de Falco et al., 2008). Esto se debe a que es un algoritmo evolutivo en el cual las variables del problema a optimizar (en este caso los parámetros de la CTRNN) están codificadas como un vector de números reales, llamado genotipo, cuya longitud es igual al número de variables del problema. Esta codificación como vector de números reales es ideal para codificar los parámetros de la CTRNN.

Debido a la técnica de selección utilizada por la Evolución Diferencial, las ventajas que presenta éste algoritmo respecto a los algoritmos genéticos tradicionales son: (1) la siguiente generación siempre tendrá un comportamiento igual o mejor que su antecesora; (2) su operador de mutación utiliza la distribución actual de los vectores en la población, lo que permite adaptar la mutación a la situación de búsqueda propia que tenga el algoritmo, lo cual parece ser una de sus principales ventajas (Prize, 1999).

Para una explicación más detallada sobre qué son y cómo funcionan en general los algoritmos genéticos puede consultarse el Anexo C.1. En el Anexo C.2 se puede encontrar una descripción más detallada del algoritmo de Evolución Diferencial.

### Configuración del algoritmo para los experimentos

La versión estándar de la Evolución Diferencial se muestra en el Algoritmo 2.1. Se deben tener en cuenta las siguientes consideraciones en cuanto a los parámetros del algoritmo:

- La cantidad máxima de generaciones $\mathbf{G_{max}}$ dependerá de la complejidad de cada experimento.
- Para que el algoritmo de Evolución Diferencial pueda asegurar cierto éxito de convergencia a una solución, el tamaño de la población **NP** debe ser, al menos, 10 veces el tamaño del genotipo (Prize, 1999).
- El parámetro **CR** controla la influencia del vector de mutación en la generación del vector hijo. Para cada individuo de la población tendrá un valor en el rango [0.2, 0.9] directamente proporcional al su valor de adaptación (el cual se explica más adelante). Valores cercanos a 0.9 implican que el vector hijo será muy parecido al vector de mutación. Por el contrario, valores cercanos a 0.2 indican que el vector hijo sera muy parecido al vector padre.
- El parámetro **F** permite escalar las diferencias entre vectores para calcular el vector de mutación. Se ha definido de forma que para cada miembro de la población se selecciona un número aleatorio en el rango [0.5, 1.0].
- La función randint(min,max) regresa un numero entero entre min y max, mientras que rand[0,1) es una función que devuelve un número real entre 0 y 1. Ambas funciones están basadas en una distribución uniforme de números aleatorios.

Debido a que los valores del genotipo son números reales que se encuentran en el rango [0,1], el resultado de la recombinación del vector padre con el vector mutación para la obtención del vector hijo se expresará en módulo 1.

### Espacio "Fitness"

La función "fitness" o función de adecuación codifica cuál de los dos individuos, el padre o el hijo, pasará a formar parte de la siguiente generación. Al final de cada experimento, se considerará como resultado del mismo a aquel individuo de la última población con mayor valor para la función "fitness". En el caso concreto de este trabajo, también determinará el valor de CR que debe aplicarse en la mutación de cada individuo para la obtención de su vector hijo. En el Anexo C.3 se expone el Espacio "Fitness" propuesto como estructura para describir, evaluar y comparar funciones de adecuación.

**Algoritmo 2.1** Algoritmo de Evolución Diferencial.

```
Begin
    G = 0
    Crear aleatoriamente la población inicial x̄_G ∀i, i = 1, .., NP
    Evaluar f(x̄_G) ∀i, i = 1, .., NP
    For G = 1 to G_max Do
        For i = 1 to NP Do
            Seleccionar aleatoriamente r1 ≠ r2 ≠ r3
            j_rand = randInt(1, D)
            For j = 1 to D Do
                If ((rand_j[0, 1) < CR) or (j = j_rand)) then
                    u^i_{j,G+1} = x^{r3}_{j,G} + F(x^{r1}_{j,G} − x^{r2}_{j,G})
                Else
                    u^i_{j,G+1} = u^i_{j,G}
                End if
            End For
            If (f(ū^i_{G+1}) ≤ f(x̄^i_G)) then
                x̄^i_{G+1} = ū^i_{G+1}
            Else
                x̄^i_{G+1} = x̄^i_G
            End if
        End For
        G = G + 1
    End For

End
```

## 2.4. Diseño de los experimentos

Para el diseño de cada uno de los experimentos se seguirán los siguientes pasos:

1. Se diseñarán mapas adecuados para la evaluación del comportamiento de cada uno de los individuos de cada generación del algoritmo genético.

2. Se diseñarán el bot según sus sensores para obtener información del entorno y sus salidas para realizar sus acciones.

3. Se diseñará el modelo de CTRNN en función su número de neuronas, las conexiones entre las mismas y la existencia de simetría bilateral, además de definir valores mínimos y máximos para cada uno de sus parámetros.

4. Se definirán los parámetros para la ejecución del algoritmo de Evolución Diferencial. Además, se diseñará una función "fitness" o de adecuación que se ajuste al comportamiento deseado para cada bot.

5. Se generará una población inicial de CTRNNs, cuyos parámetros se establecerán alatoriamente, y se harán evolucionar dichos parámetros mediante el algoritmo genético de Evolución Diferencial.

6. Al finalizar la ejecución del algoritmo de Evolución Diferencial, se considerará como solución al experimento el bot controlado por la CTRNN que haya obtenido mayor valor "fitness" durante el experimento.

# Capítulo 3

# Aprendizaje evolutivo para la obtención de CTRNNs con comportamientos de navegación

Este capítulo está dedicado a la utilización del aprendizaje evolutivo para obtener dos bots controlados por CTRNNs en UT2004, las cuales aprovechen la capacidad de memoria a corto plazo proporcionada por las recurrencias entre los nodos de la red. Para ello se utilizará un aprendizaje basado en algoritmos genéticos, concretamente el de Evolución Diferencial (Prize, 1999). El objetivo es que, gracias al proceso de evolución, emerjan en las CTRNNs comportamientos de navegación que no serían posibles utilizando redes neuronales feed-forward. Para ello, se realizarán dos experimentos. El primero de ellos está orientado a la obtención de una CTRNN como controlador para un bot con comportamiento de navegación y evitación de obstáculos en el entorno no estructurado de UT2004. En el segundo, se busca obtener una CTRNN capaz de seguir la trayectoria de movimiento de un bot enemigo, incluso cuando lo pierde momentáneamente de vista al desaparecer éste tras un muro, para lo cual tendrá que poder "predecir" su reaparición. La sección 3.1 muestra la configuración general para los dos experimentos. Las secciones 3.2 y 3.3 están dedicadas al primer y segundo experimento respectivamente. Por último, la sección 3.4 recoge las conclusiones extraídas de ambos experimentos.

## 3.1. Configuración general de los experimentos

Dado que ambos experimentos comparten ciertos aspectos de su configuración, éstos se recogen a continuación. Las características específicas de cada experimento se comentarán en la sección dedicada a cada uno de ellos.

### 3.1.1. Diseño del bot

El diseño del bot, es decir, su capacidad de percibir su entorno y de ejecutar sus acciones, viene limitado por las posibilidades del entorno de simulación utilizado. El controlador que se obtenga mediante el proceso de evolución para el bot dependerá de la situación o entorno en el que se desarrolle y de cómo haya sido diseñado el agente (Izquierdo, 2008).

En el caso de UT2004 utilizando *Pogamut*, es posible diseñar los sensores del bot mediante un sistema de "raytracing", lo que permite diseñar nuestro bot según la arquitectura de un robot *Khepera* (Floreano y Mondada, 1994), cuya estructura a su vez está basada en la arquitectura de un Vehículo de *Braitenberg* (Braitenberg, 1984). Se considera pues que el bot tiene una forma circular, un sistema de "raytracing" para obtener información del entorno y dos motores diametralmente

opuestos que permitirán al bot desplazarse y girar. Dada la naturaleza de los experimentos, se han eliminado los sensores traseros del bot, y la distribución y longitud de los seis sensores frontales se adaptará según convenga en cada uno de ellos. En la Figura 3.1 podemos ver la arquitectura de un Vehículo de *Braitenberg* (Braitenberg, 1984), así como el robot *Khepera* y un bot en UT2004 basados en dicha arquitectura.

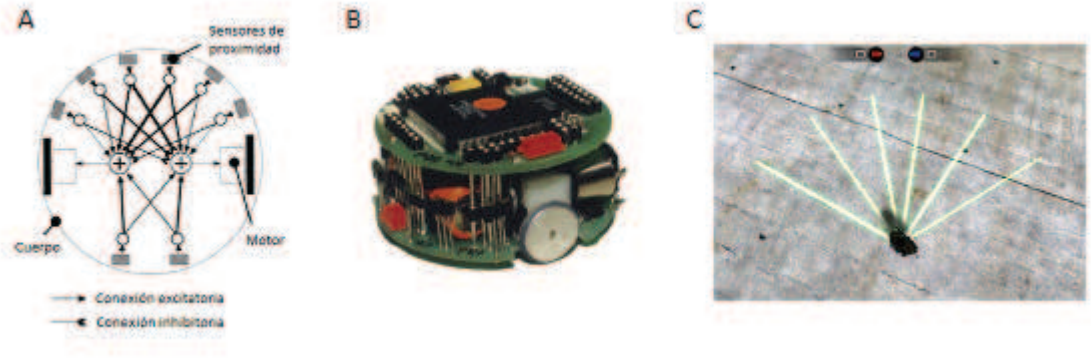



Figura 3.1: Vehículo de Braitenberg y su versión real y simulada. [A] Representación esquemática del un Vehículo de Braitenberg. [B] Robot Khepera, cuya arquitectura es similar a un Vehículo de Braitenberg. [C] Bot de UT2004 con arquitectura similar a un Vehículo de Braitenberg.

El funcionamiento de los sensores consistirá en devolver un número real de doble precisión entre 0 y 1, dependiendo de la cercanía o lejanía de un obstáculo (en el caso del experimento 1) u objetivo (en el caso del experimento 2): si el sensor no detecta ningún obstáculo (u objetivo), devolverá 0; en caso de detectar algo, el valor devuelto aumentará linealmente cuanto más cerca se encuentre el obstáculo (u objetivo) del bot.

En cuanto a la ejecución de las acciones de giro y movimiento por parte del bot, éstas vendrán condicionadas por la actividad de los motores (los cuales reciben un valor entre [0,1]), cada uno de los cuales está conectado a la salida de una de las neuronas de salida de la red neuronal. Parámetros tales como la velocidad máxima de movimiento y el ángulo de giro máximo son propios de cada experimento.

### 3.1.2. Configuración de las CTRNNs

Como ya se ha indicado en la sección 2.2, para los experimentos se van a utilizar CTRNNs en las que todas las neuronas estén totalmente interconectadas, autoconectadas, conectadas a su vez a todos los sensores y dos de las neuronas están conectadas a las salidas de los motores. Se buscará obtener la redes neuronales lo más pequeñas posibles que sean capaces de adaptarse satisfactoriamente al comportamiento deseado.

Así pues, la CTRNN más pequeña posible para el bot con seis sensores que se va a utilizar en los experimentos, sería una red simétrica con dos neuronas interconectadas y autoconectadas. Si debido a su sencillez no fuera posible encontrar una solución óptima al problema, se podría introducir una o varias neuronas, también totalmente interconectadas y autoconectadas, cuya salida no estaría conectada a ningún motor.

El modelo de la neurona para estos experimentos está basado en la ecuación de la CTRNN 2.1 descrita en la sección 2.2. En simulación, la activación de los nodos se calcula a través del tiempo usando la integración de Euler con un tiempo de ciclo $\Delta t$, con lo que se obtiene la ecuación

$$y_i(n+1) = y_i(n) + \frac{\Delta t}{\tau_i} * \left( -y_i(n) + \sum_{j=1}^{M} w_{ji} * \sigma(y_j - \theta_j) + \sum_{k=0}^{N} u_{ki} I_k \right) \qquad i = 1, 2, \ldots, M \quad (3.1)$$

donde $i$ es un índice ($i = 1, 2, ..., M$) y $M$ es el número de neuronas, $N$ es el número de entradas de la red, $y_i$ es el estado actual de la neurona, $\Delta t$ es el tiempo de ciclo para la integración, $\tau_i$ es la constante temporal de activación de la neurona, $w_{ji}$ es el peso de la conexión entre las neuronas $i$ y $j$, $\theta$ es el término bias, $I_k$ representa la entrada $k$ y $\sigma(x)$ es la función sigmoidal. Debido al tiempo de ciclo proporcionado por *GameBots* indicado en la sección 2.1, el valor de $\Delta t$ se ha establecido en 0.4 segundos.

En la tabla 3.1 se muestran los valores mínimos y máximos para los parámetros de las CTRNNs usadas en los experimentos.

| **Parámetros CTRNNs** | | |
|---|---|---|
| $\tau$ | $\theta$ | $\omega$ |
| [$e^0$, $e^4$] | [-2.0, 2.0] | [-5.0, 5.0] |

Tabla 3.1: Valores mínimos y máximos para los parámetros de las CTRNNs de los experimentos

### 3.1.3. Configuración del algoritmo de Evolución Diferencial

#### Codificación de los genotipos

El genotipo, en el cual los parámetros de la red neuronal están codificados, consiste en un vector de números reales de doble precisión (genes) cuyos valores se encuentran en el rango [0,1]. Como resultado de la simetría bilateral de la CTRNN, el tamaño del genotipo se reduce, ya que dos neuronas simétricas se codifican juntas.

El genotipo correspondiente al modelo más pequeño de CTRNN totalmente conectada e interconectada, se obtiene a partir del cromosoma de la Tabla 3.2. La decodificación del genotipo para obtener los valores que definen la CTRNN se realiza de la siguiente manera:

- El valor de $\tau$ se obtiene elevando el número e al valor del gen 0.
- Los valores bias ($\theta$), los pesos de las conexiones entre neuronas de entrada y salida (u), y los pesos de las conexiones entre las neuronas de salida (w), reciben un valor entre su mínimo y máximo establecidos, directamente proporcional al valor del gen correspondiente.

| **Cromosoma neuronas de salida 0 y 1** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $\tau$ | $\theta$ | u | | | | | | w | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Tabla 3.2: Codificación del genotipo de una CTRNN totalmente interconectada, autoconectada y simétrica con 6 neuronas de entrada y 2 de salida. Los genes correspondientes a los pesos de las conexiones han sido codificados de forma ordenada, de modo que el gen 2 corresponde a la conexión u10 y (debido a la simetría) a la u25, el gen 3 a las conexiones u11 y u24, y así sucesivamente. En cuanto a los pesos w, el gen 8 corresponde a las autoconexiónes y el 9 a las interconexiones entre las neuronas de salida.

## 3.2. Experimento 1: Navegación y evitación de obstáculos en un entorno no estructurado

Desde el punto de vista de la navegación en el entorno, en los videojuegos comerciales pueden ocurrir situaciones no predichas y, por consiguiente, no especificadas por el programador, frente a las cuales el personaje es incapaz de reaccionar. Debido a ello, en ocasiones se pueden encontrar personajes corriendo contra la pared, atascados en una esquina, dando vueltas sobre sí mismos, etc. Mediante la aplicación de las técnicas presentadas a lo largo de este proyecto, se pretende obtener bots que eviten este tipo de comportamientos indeseables.

### 3.2.1. Descripción del problema

En este primer experimento se busca obtener una CTRNN como controlador para un bot con comportamiento de navegación y evitación de obstáculos en el entorno no estructurado de UT2004 (se entiende como entorno no estructurado a un entorno en el que no es viable que un agente pueda disponer de un mapa por lo complejo o lo cambiante del mismo (Arkin, 1998))

Este tipo de experimento ya ha sido realizado anteriormente para entornos diferentes a los videojuegos por otros autores con resultados satisfactorios. Floreano y Mondada (1996) utilizaron un robot *Khepera* real, el cual, tras el proceso de evolución, era capaz de desplazarse por el circuito sin colisionar contra sus muros, para lo cual era capaz de regular su velocidad e incluso dar marcha atrás. Por otro lado, Bourquin (2004) utilizó un entorno simulación que emulaba batallas entre tanques que eran capaces de desplazarse evadiendo obstáculos, para lo que programó manualmente el detener el movimiento de los tanques en caso de peligro de colisión, en lugar de dejar esta decisión en manos de la red neuronal. Ambos utilizaron redes de Elman en sus respectivos trabajos.

A diferencia que en estos trabajos, se pretende que la CTRNN resultante de este experimento permita al bot moverse con un desplazamiento siempre hacia adelante y velocidad constante (sin dar marcha atrás como ocurría en (Floreano y Mondada, 1996)) y no se detendrá al bot en caso de peligro de colisión (lo que no ocurría en (Bourquin, 2004)). Además, en lugar de redes de Elman se utilizarán CTRNNs lo más pequeñas posibles con sus neuronas totalmente interconectadas y autoconectadas.

La Figura 3.2 (ver Nolfi y Floreano, 2000, pág. 78), es un claro ejemplo de las ventajas que ofrece la evolución de robots utilizando CTRNNs frente a la evolución de arquitecturas feed-forward.

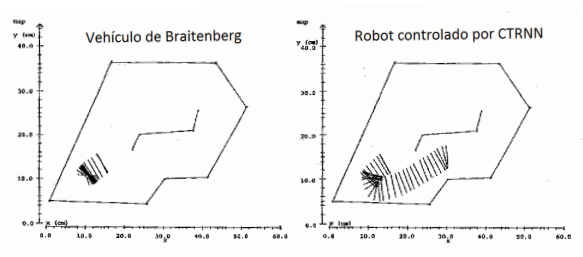



Figura 3.2: Comparación entre una arquitectura feed-forward y otra con conexiones recursivas. **Izquierda:** Un sistema de control basado en un vehículo de Braitenberg con conexiones feedforward simétricas se mueve hasta la esquina inferior izquierda, donde se detiene al encontrar las mismas intensidades en los sensores de ambos lados (las pequeñas oscilaciones se deben al ruido sensorial). **Derecha:** El controlador evolucionado hace uso de la recursividad para evitar el punto muerto.

### 3.2.2. Diseño del experimento

#### Diseño del mapa para la simulación

El entorno de simulación diseñado corresponde al del mapa de la figura 3.3. Como puede apreciarse, se trata de un entorno con obstáculos en el que el bot deberá ser capaz de moverse sin colisionar.

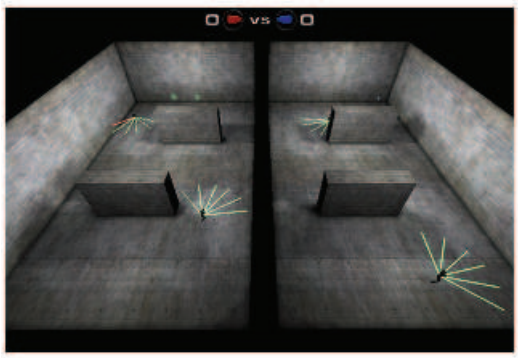


Figura 3.3: Mapa para el experimento para la obtención de un bot controlado por una CTRNN con capacidad de navegación y evitación de obstáculos.

#### Función "Fitness"

A la hora de elegir una función "fitness que defina el comportamiento deseado en este experimento, no será suficiente con que ésta premie únicamente el comportamiento de desplazarse sin colisionar contra los obstáculos que encuentre en su camino. Un bot que diera vueltas sobre sí mismo cumpliría perfectamente con esta descripción, pese a ser éste un comportamiento totalmente indeseable. Por tanto, el diseño de este experimento deberá contar con una función "fitness" que premie tanto el comportamiento de evitación de obstáculos como el de desplazarse por su entorno sin dar vueltas sobre su propio eje.

Por esta razón, se ha elegido la función

$$\phi = (1 - \sqrt{\triangle v})(1 - i) \tag{3.2}$$

$$0 \leq \triangle v \leq 1$$

$$0 \leq i \leq 1$$

donde $\triangle v$ es el valor absoluto de la diferencia algebraica entre los valores de velocidad de los motores, e $i$ es el mayor de los valores devueltos por los sensores. El factor $(1 - \sqrt{\triangle v})$ premia que el bot se desplace en línea recta, ya que devolverá valores cercanos a 1 cuanto menor sea el giro y cercanos a 0 en casos de que el giro sea mayor. En cuanto al factor $(1 - i)$, éste premia que el bot se mantenga lo más alejado posible de los obstáculos del mapa. Esta función se evalúa cada ciclo, devolviendo un valor entre [0,1], y el resultado será la suma de todos ellos dividido por el número total de ciclos.

#### Configuración de los parámetros para el experimento

La tabla 3.3 recoge los parámetros para la configuración del experimento. Tomando como referencia los bots de ejemplo proporcionados por Pogamut, se ha definido la longitud de los rayos que componen el sistema de sensores "raytracing" como 10 veces el área de colisión del bot (en las unidades de medida utilizadas por UT2004).

| CONFIGURACIÓN EXPERIMENTO 1 | | | | | |
|---|---|---|---|---|---|
| CTRNN | | ALG. GENÉTICO | | RAYTRACING | |
| Neuronas | 6 Entrada | Tamaño Genotipo | 10 | Sensor 0 | -60º |
| | 2 Salida | Tamaño Población | 100 | Sensor 1 | -30º |
| Simetría | Si | Num. Generaciones | 75 | Sensor 2 | -10º |
| BOT | | Evaluaciones Simultáneas | 4 | Sensor 3 | 10º |
| Ángulo de giro máximo | 45º | Evaluaciones por bot | 3 | Sensor 4 | 30º |
| Velocidad Máxima | 0.5 | Tiempo Evaluación | 20 seg. | Sensor 5 | 60º |

Tabla 3.3: Configuración del experimento para el primer bot

### 3.2.3. Resultados del experimento

En la gráfica de la figura 3.4 se pueden ver los valores devueltos por la función "fitness" durante el proceso de evolución. El bot que mejor se adapta a la tarea lo hace con un valor "fitness" de 58,7 %, y sus parámetros se muestran en la figura 3.5. El hecho de no haber obtenido un bot con un valor de adaptación cercano al 100 % se debe a que la función "fitness" elegida penaliza situaciones inevitables para el bot, como la de girar o que los sensores detecten algún obstáculo.

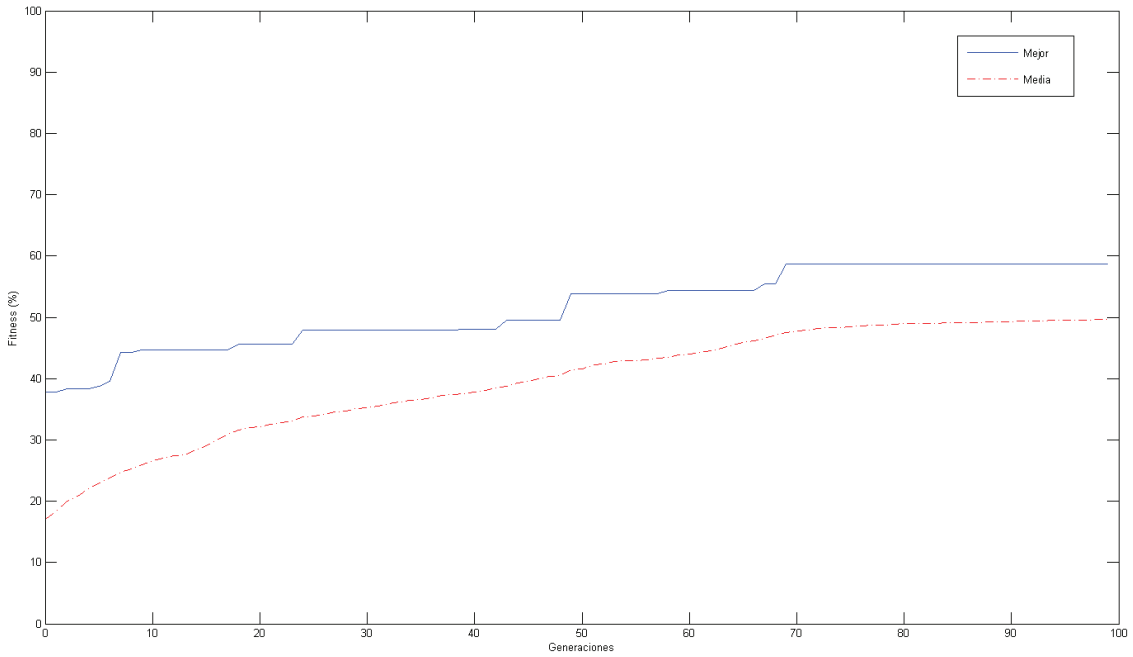
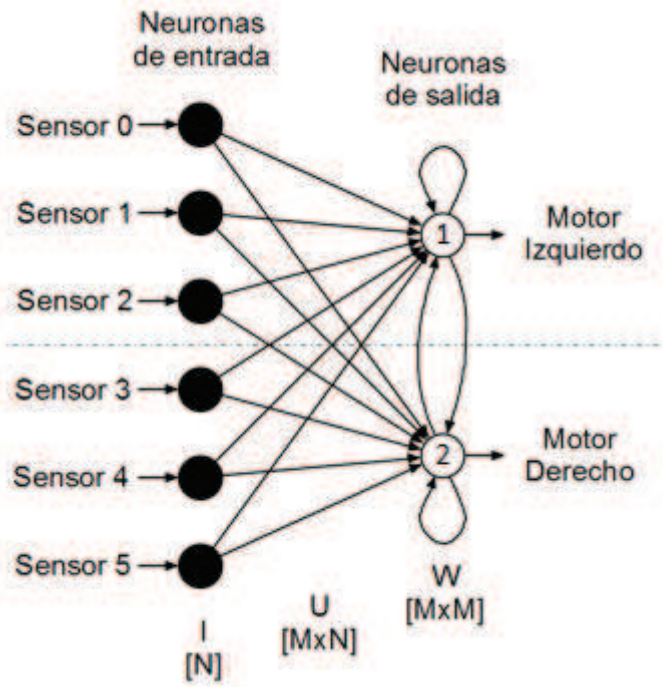



Figura 3.4: Gráfica con los resultados de la función "fitness" obtenidos durante el proceso de evolución para la obtención de una CTRNN con capacidad de navegación y evitación de obstáculos. La línea azul muestra el valor obtenido por el mejor individuo de cada generación. La línea discontinua roja muestra la media de todos los individuos de cada generación.

En la Figura 3.6 se muestra un ejemplo gráfico del comportamiento del bot. El hecho de haber introducido un pequeño error en las salidas de la red neuronal hace que el bot no camine totalmente en línea recta, sino que le permite dar pequeños giros que le proporcionan cierta capacidad de exploración del entorno.




| | y1 | y2 |
|---|---|---|
| $\tau$ | 1,0100 | 1,0100 |
| $\theta$ | 0,9817 | 0,9817 |
| s0 | -3,6816 | 4,8518 |
| s1 | -4,6967 | 4,6429 |
| s2 | -2,6003 | 1,0035 |
| s3 | 1,0035 | -2,6003 |
| s4 | 4,6429 | -4,6967 |
| s5 | 4,8518 | -3,6816 |
| y1 | -2,1309 | 4,2978 |
| y2 | 4,2978 | -2,1309 |

Figura 3.5: CTRNN con comportamiento de navegación y evitación de obstáculos en un entorno no estructurado para un bot con estructura de robot Khepera adaptada.


Figura 3.6: Ejemplo gráfico del comportamiento de un bot controlado por la CTRNN resultado del experimento 1 para el modelo 1. [A] El bot localiza un obstáculo. [B] El bot reacciona y evita el obstáculo. [C] El bot ha evitado correctamente el obstáculo sin colisionar contra él.

### 3.2.4. Análisis del comportamiento del bot

Por último, se desea comprobar el correcto funcionamiento del bot. Además se desea comprobar si es capaz de solucionar el problema de evitar bloquearse en puntos muertos descrito en la figura 3.2, a pesar de no haber sido entrenado para ello, para lo cual el bot será capaz de regular su velocidad según el valor de la más pequeña de sus salidas, llegando incluso a detenerse si el valor de ésta es menor de 0.1 (velocidad mínima del bot en *Pogamut*), comportamiento para el cual cual tampoco ha sido entrenado. Para ello, se ha diseñado el mapa de la figura 3.7.

Al ejecutar el bot, el primer obstáculo que éste encontrará será el punto muerto de una de las esquinas. Se considerará como éxito que el bot no colisione con ningún obstáculo y sea capaz de evitar todos los obstáculos utilizando para ello menos de 15 ciclos de ejecución.

Tras ejecutar el bot 100 veces, se ha obtenido un éxito del 100 %. Esto nos demuestra no solo el éxito del experimento a la hora de obtener un bot con el comportamiento de navegación y evitación de obstáculos, sino también la flexibilidad por parte de las CTRNNs de adaptarse a situaciones y comportamientos para los cuales no han sido específicamente entrenadas.

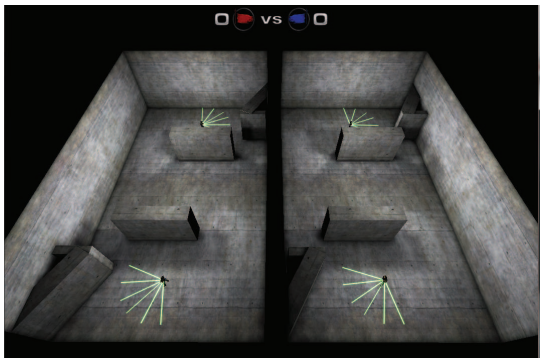



Figura 3.7: Mapa para la comprobación del correcto funcionamiento de un bot controlado por una CTRNN con capacidad de navegación y evitación de obstáculos, el cual además sea capaz de evitar puntos muertos.

## 3.3. Experimento 2: Seguimiento de trayectorias

### 3.3.1. Descripción del problema

En este segundo experimento se busca obtener una CTRNN como controlador para un bot con capacidad de seguir la trayectoria de movimiento de un bot enemigo. Para ello, el bot deberá ser capaz de girar sobre sí mismo siguiendo la trayectoria de otro bot que gire en círculos alrededor suyo.

La ventaja que ofrecen las CTRNNs como controlador del bot para esta tarea, frente a la utilización de redes neuronales feed-forward, es que, gracias su capacidad de memoria a corto plazo, será posible para el bot seguir la trayectoria de otros bots incluso cuando pierda momentáneamente el contacto visual con ellos. Un claro ejemplo de esto se produciría en condiciones del juego en que el bot al cual se está siguiendo desaparezca tras un muro para reaparecer al otro lado del mismo. El bot controlado por la CTRNN debe ser capaz de, si tal cosa sucede, seguir girando y reencontrar a su objetivo, lo cual no sería posible para una red totalmente reactiva.

### 3.3.2. Diseño del experimento

#### Evolución incremental

En este experimento, la evolución se realizará de manera incremental en dos fases:

1. En la primera fase, se buscará que la CTRNN sea capaz de controlar un bot con capacidad de enfocar a otro bot "objetivo" que permanecerá estático en todo momento, de forma que éste quede entre los dos rayos internos de su sistema "raytracing". Para ello, el bot permanecerá fijo en su lugar de origen y sólo será capaz de girar sobre sí mismo. Al comienzo de la ejecución del bot, éste podrá sentir al bot "objetivo" por uno de sus sensores externos.

2. En la segunda fase se realiza a partir de la población obtenida tras la primera. Al principio de la ejecución, el bot tiene localizado al bot "objetivo" entre sus dos sensores internos. La CTRNN debe ser capaz de seguir su trayectoria de movimiento. El bot al que se evalúa sólo será capaz de girar sobre sí mismo, al igual que en la primera fase, mientras que el bot "objetivo" girará en círculos alrededor de él, de forma que habrá momentos en los que se ocultará al pasar por detrás de algún muro. El bot debe ser capaz de seguir su trayectoria pese a perder el contacto visual durante algunos ciclos.

### Diseño de mapas para la simulación

Para este experimento se ha diseñado el mapa que podemos ver en la Figura 3.3, el cual se ha diseñado por duplicado para poder así realizar varias evaluaciones en paralelo. En la segunda fase del experimento, el bot "objetivo" girará en círculos alrededor del bot que se desea entrenar, movimiento que le lleva a desaparecer tras los muros dos veces por cada vuelta.

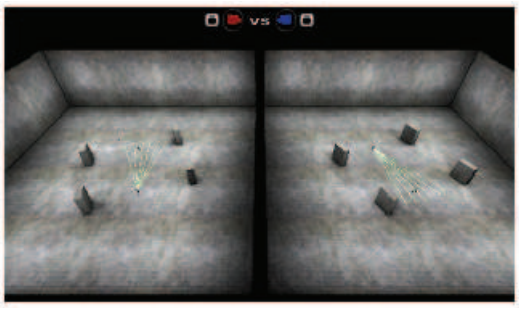



Figura 3.8: Mapa para el segundo experimento.

### Adaptación del sistema "Raytracing"

Debido al tamaño del bot "objetivo", es muy difícil que los rayos que componen el sistema "raytracing" del bot controlado por la CTRNN impacten directamente en él para localizar su posición. Debido a ello, gracias a las ventajas que ofrece trabajar en un sistema virtual de simulación, se simula un agrandamiento del bot "objetivo". Para ello, se ha adaptado el sistema "raytracing" de forma que el sensor se activa si el bot "objetivo" se encuentra a $3^{\underline{o}}$ o menos del ángulo formado por el bot controlado por la CTRNN como vértice, el rayo del "raytracing" y la línea que une ambos bots. Para valores mayores a $3^{\underline{o}}$ se considera que el bot se encuentra entre dos rayos, por lo que ambos se activan.

Debe tenerse en cuenta que el controlador del bot es una red simétrica, por lo que en caso de que todos sus sensores devuelvan 0 (es decir, ningún bot "objetivo" se encuentre en su campo de visión), sus motores realizarán la misma acción en sentidos contrarios, provocando como resultado que el bot no realice acción alguna. Para evitar esto, se le añade un pequeño valor aleatorio a la entrada de cada sensor, lo cual le permite girar en uno u otro sentido en busca de otros bots.

### Función "Fitness"

La función "fitness" elegida para este experimento es la función

$$\phi = \frac{\sum_{n=1}^{numCiclos} \varphi(n)}{numCiclos} \tag{3.3}$$

$$\text{siendo } \varphi(n) = \sum_{i=1}^{N} I_i^{'} r_i, \text{ donde } I_i^{'} = \begin{cases} 1 & I_i^{'} > 0 \\ 0 & otros\, casos \end{cases}$$

y donde $N$ es el número de sensores, $I_i^{'}$ es el valor de activación del sensor $i$, y $r_i$ es el valor recompensa del sensor $i$, definido como [0.1, 0.3, 1.0, 1.0, 0.3, 0.1], donde los valores más grandes corresponden a los sensores centrales y los más pequeños a los exteriores.

Para esta función "fitness", la función $\varphi(n)$ se calcula en cada ciclo, y devuelve un valor proporcional al sensor o sensores que están localizando al bot objetivo en ese instante, en términos del vector de recompensas. Una vez terminada la evaluación del individuo, la suma de todas las

recompensas se dividirá por el número de ciclos totales, obteniéndose así el porcentaje de la eficiencia a la hora de seguir la trayectoria del bot manteniendo durante el mayor tiempo posible al bot enemigo en el punto de mira.

**Configuración del experimento**

La tabla 3.4 recoge los parámetros para la configuración del experimento. La longitud de los rayos es 50 veces el área de colisión del bot (en las unidades de medidas utilizadas por UT2004). Los dos valores para el parámetro "Evaluaciones Simultáneas" corresponden a la fase 1 y 2 respectivamente.

| CONFIGURACIÓN EXPERIMENTO 1 | | | | | |
|---|---|---|---|---|---|
| CTRNN | | ALG. GENÉTICO | | RAYTRACING | |
| Neuronas | 6 Entrada | Tamaño Genotipo | 10 | Sensor 0 | -17º |
| | 2 Salida | Tamaño Población | 100 | Sensor 1 | -10º |
| Simetría | Si | Num. Generaciones | 55 | Sensor 2 | -3º |
| BOT | | Evaluaciones Simultáneas | 4 / 2 | Sensor 3 | 3º |
| Ángulo de giro máximo | 5º | Evaluaciones por bot | 2 | Sensor 4 | 10º |
| Velocidad Bot Objetivo | 0,4 | Tiempo Evaluación | 30 seg. | Sensor 5 | 17º |

Tabla 3.4: Configuración del experimento para el segundo bot

### 3.3.3. Resultados

En la gráfica de la figura 3.9 se pueden ver los valores devueltos por la función "fitness" durante el proceso de evolución.

Tras la ejecución de la primera fase del experimento durante las 10 primeras generaciones, se ha obtenido un bot con un valor de adaptación de 99.61 %. Como se puede apreciar, tan sólo han sido necesarias dos generaciones para alcanzado dicho valor. En la Figura 3.10 puede verse un ejemplo gráfico del comportamiento obtenido. Para asegurar el correcto funcionamiento de la CTRNN obtenida, se ejecutó el bot 100 veces durante 20 segundos cada una, considerando como éxito que al final de la ejecución del bot, el bot "objetivo" estuviese entre los dos rayos centrales del sistema "raytracing". El bot tuvo éxito en la tarea un 100 % de las veces, éxito tras el cual se decidió comenzar la segunda fase del experimento.

Tras la ejecución de la segunda fase del experimento hasta llegar a 55 generaciones, se ha obtenido un bot con un valor de adaptación de 78.71 %. La reducción de éste valor se debe a que en esta ocasión, el bot "objetivo" se encuentra en movimiento, por lo que habrá ciclos en los que el bot controlado por la CTRNN lo localizará por sus rayos intermedios o incluso externos, o incluso por ninguno en caso de que desaparezca tras un muro. Además, en la gráfica de la figura 3.9 se aprecia un descenso del valor "fitness" en la transición de la fase 1 a la 2, ya que la CTRNN tiene que adaptarse al nuevo comportamiento.

La CTRNN obtenida como resultado tras el experimento genético es la mostrada en la Figura 3.11. En la Figura 3.12 podemos ver un ejemplo gráfico de su capacidad de seguir a un objetivo incluso cuando éste desaparece tras un muro.

Para asegurar el correcto funcionamiento de la CTRNN, se ejecutó el bot 100 veces durante 30 segundos cada una, considerando como fracaso el hecho de que el bot no perdiese a su objetivo durante más de 15 ciclos de ejecución. El bot tuvo éxito en la tarea en un 92 % de las veces. Tras observar su comportamiento, se llegó a la conclusión de el 8 % de ejecuciones no exitosas venía provocado por el hecho de que justo antes de que el bot "objetivo" desaparezca tras el muro, el bot controlado por la CTRNN se adelanta a su trayectoria de movimiento y lo localiza con la parte del "raytracing" que le hace girar en sentido contrario a la trayectoria de movimiento del bot para esperarlo.

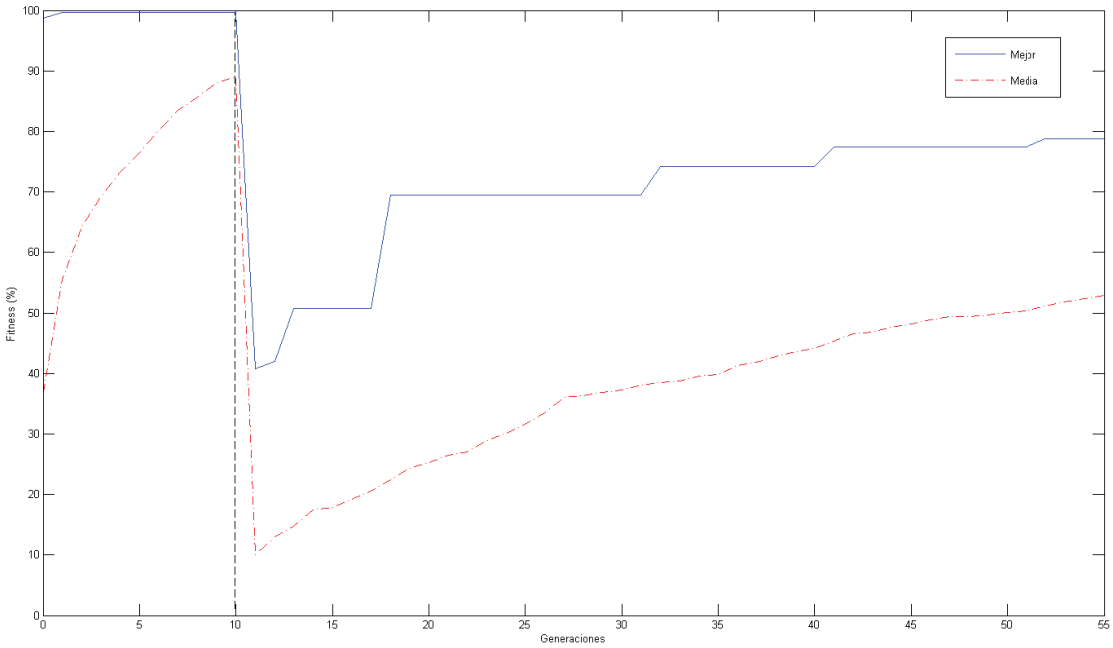



Figura 3.9: Gráfica con los resultados de la función "fitness" obtenidos tras el proceso de evolución para la obtención de una CTRNN con capacidad de seguimiento de las trayectorias de movimiento de otros bots. La línea azul muestra el valor obtenido por el mejor individuo de cada generación. La línea discontinua roja muestra la media de todos los individuos de cada generación. La línea discontinua vertical muestra la transición de la primera fase (fase de localización) a la segunda (fase de seguimiento).

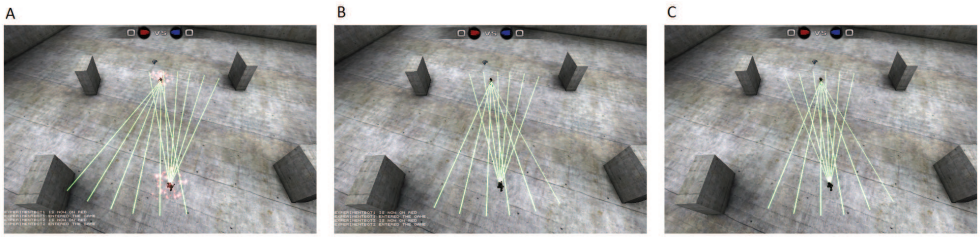
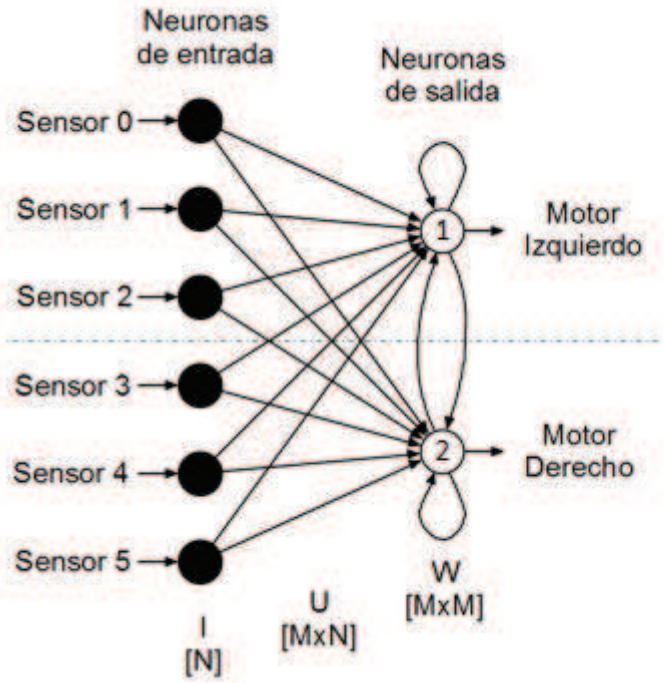



Figura 3.10: Ejemplo gráfico del comportamiento de dos bots controlados por la CTRNN resultado de la primera fase del experimento 2.

## 3.4. Conclusiones: memoria a corto plazo en CTRNNs con recurrencias entre sus nodos

Como se ha podido ver, las recurrencias entre los nodos de la CTRNN dotan a ésta de una memoria a corto plazo que permite realizar tareas que serían imposibles para una red neuronal feed-forward, incluso cuando se utilizan CTRNNs simétricas tan sencillas como las utilizadas en éste capítulo, con solo dos neuronas totalmente autoconectadas e interconectadas. En el primer experimento se ha podido comprobar cómo un bot con estructura de robot *Khepera* adaptada y controlado por una CTRNN obtenida mediante aprendizaje evolutivo, es capaz de un comportamiento de navegación y evitación de obstáculos. Además, se ha comprobado que, a pesar de no haber sido entrenado explícitamente para ello, es capaz de regular su velocidad y evitar puntos




| | y1 | y2 |
|---|---|---|
| $\tau$ | 1,3764 | 1,3764 |
| $\theta$ | -0,4462 | -0,4462 |
| s0 | 1,1450 | -1,3610 |
| s1 | -4,2409 | 4,1777 |
| s2 | -2,2718 | -1,1175 |
| s3 | -1,1175 | -2,2718 |
| s4 | 4,1777 | -4,2409 |
| s5 | -1,3610 | 1,1450 |
| y1 | -3,7736 | -3,5014 |
| y2 | -3,5014 | -3,7736 |

Figura 3.11: CTRNN con comportamiento de seguimiento de trayectorias de movimiento de otros bots para un bot con estructura de robot Khepera adaptada.


Figura 3.12: Ejemplo gráfico del comportamiento de un bot controlado por la CTRNN resultado del experimento 2. [A] El bot sigue la trayectoria de su objetivo. [B] El objetivo se oculta tras un muro. [C] El objetivo reaparece y el bot ha sido capaz de seguir su trayectoria.

muertos sin quedarse bloqueada, lo cual a su vez muestra la flexibilidad de este tipo de redes para adaptarse a situaciones y comportamientos para los cuales no han sido específicamente entrenadas. En el segundo experimento se ha obtenido con éxito mediante aprendizaje evolutivo un bot controlado por una CTRNN, cuyas recurrencias entre sus nodos le permitían seguir la trayectoria del bot "objetivo" incluso cuando éste desaparecía tras un muro, para lo cual tenía en cuenta su comportamiento anterior. Una vez estudiado el efecto que tienen en el comportamiento de las CTRNNs las recurrencias entre sus nodos, en el siguiente capítulo se estudiará su capacidad de aprendizaje en el tiempo de ejecución del bot cuando se obtienen CTRNNs cuyas neuronas presentan actividad multiescalada en el tiempo.

En cuanto al entorno de simulación UT2004, los experimentos realizados muestran la posibilidad de aplicar la búsqueda de soluciones mediante el aprendizaje evolutivo en el videojuego utilizando *Pogamut* para la obtención de CTRNNs con pocos parámetros.

# Capítulo 4

# Aprendizaje en CTRNNs sin plasticidad sináptica durante el tiempo de vida del bot

El objetivo de este capítulo es el de obtener un bot controlado por una CTRNN sin plasticidad sináptica con capacidad de aprendizaje en tiempo real, es decir, el bot será capaz de aprender sin realizar cambios en los parámetros de la red. En la sección 4.1 se describe el experimento y el comportamiento deseado para el bot. La sección 4.2 está dedicada a la obtención mediante evolución de la CTRNN. En la sección 4.3 se analiza la capacidad de la CTRNN obtenida de aprender sin plasticidad sináptica. En la sección 4.4 se realiza un análisis de las dinámicas de dicha CTRNN con su entorno para analizar de manera formal y en profundidad el comportamiento del bot a partir de sus diagramas de bifurcación. Por último, en la sección 4.5 se muestran las conclusiones extraídas del experimento.

## 4.1. Descripción del experimento

A la hora de elegir un determinado comportamiento para el bot, en el cual se requiera aprendizaje durante su tiempo de ejecución, se ha optado por adaptar al entorno UT2004 el comportamiento mostrado por el nematodo *Caenorhabditis elegans* (Hedgecock y Rusell,1975). Dicho comportamiento consiste en asociar dos estímulos (aprendizaje asociativo): temperatura y comida. La elección de este modelo se debe a que el *C. elegans* es una elección muy común entre los investigadores en el área de la evolución de CTRNNs, ya que muestra comportamientos lo suficientemente sencillos como para poder ser modelados por CTRNNs pequeñas y suficientemente complejos como para explotar las capacidades de memoria de las mismas (Izquierdo, 2008).

A la hora de adaptar este modelo al entorno UT2004 utilizando Pogamut, se utiliza un entorno 2D con un gradiente de altura a lo largo de una de sus dimensiones, el cual se muestra en la figura 4.1-A. En él habrá 2 tipos de base: “enemiga” y “aliada”. Cada base puede encontrarse solo en regiones con un rango particular de alturas: “alta” entre [9,10] y “baja” entre [-10,-9]. La región en la que se encuentra cada base depende del tipo de entorno: en el A-ent la base “enemiga” se encuentra en la región “alta” y la “aliada” en la “baja”, mientras que en el B-ent la base “enemiga” se encuentra en la región “baja” y la “aliada” en la “baja”. El gradiente de altura se extiende por todo el entorno, el cual está libre de obstáculos. Para ello, se ha diseñado un entorno como el que se muestra en la figura 4.1-B.

Se pretende que el bot sea capaz de asociar durante su ejecución altura y base “enemiga” en cada uno de los entornos descritos y memorizarla para volver a dicha altura en caso de ser reubicado en el centro del mapa. Para ello, el bot aparecerá en la posición 0 del gradiente de altura en una

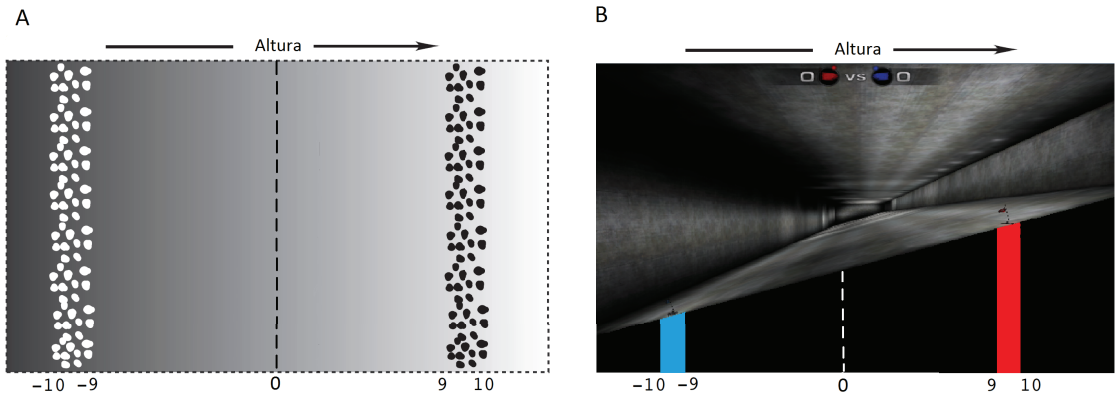



Figura 4.1: Entorno de simulación para el experimento. (A) Entorno de simulación teórico bidimensional, con un gradiente de alturas, en el que la base "enemiga" puede ser localizada en una de las dos franjas representadas por regiones a puntos. (B) Entorno de simulación en UT2004, en el que el gradiente es la altura a la que se encuentra el bot, y las franjas roja y azul representan dónde se encuentran las bases "alta" y "baja" respectivamente.

orientación aletoria. A partir de ese momento, el bot contará con 200 ciclos de ejecución para desplazarse por todo el entorno en busca de la base "enemiga" y permanecer en esa región lo más eficientemente posible. Una vez pasados los 200 ciclos, el bot se vuelve a situar en la altura 0 con una orientación aleatoria, y debe ser capaz de subir o bajar en el gradiente de altura dependiendo de si en la ejecución anterior aprendió que estaba en un entorno A-ent o B-ent. En caso de que se cambie el tipo de entorno, el bot tiene que ser capaz de reaprender y cambiar su preferencia de altura.

El bot, al igual que en los experimentos del capítulo anterior, se considera como un cuerpo redondo con dos motores diametralmente opuestos, solo que esta vez tiene únicamente dos sensores:

- Los motores permiten al agente moverse hacia adelante y girar. La velocidad de cada uno de ellos depende directamente de la el valor devuelto por una de las neuronas de salida (un valor entre [0,1)). Por ello, la velocidad de movimiento será directamente proporcional a la salida de la neurona con un valor menor (un valor entre 0 y la velocidad máxima del bot). En cuanto al giro, éste será proporcional a la diferencia entre el valor del motor derecho y el izquierdo (un valor negativo produciría un giro a la izquierda y un valor positivo a la derecha) y está acotado por un valor máximo de giro.

- El sensor de altura puede tener cualquier valor real.

- El sensor de "base" devuelve un valor 0 a no ser que el bot se encuentre en una de las bases: devolverá B=1 si se encuentra en la base "enemiga", y B=-1 si se encuentra en la base "aliada".

## 4.2. Aprendizaje evolutivo para la obtención de la CTRNN

El modelo utilizado en este experimento es el de una CTRNN con todas sus neuronas totalmente conectadas e interconectadas. Así pues, dado que el modelo de la neurona para este experimento está basado en la ecuación de la CTRNN 2.1 descrita en la sección 2.2, al aplicar la integración de Euler con un tiempo de ciclo $\Delta t$ para la activación de los nodos en simulación, se obtiene la ecuación

$$y_i(n+1) = y_i(n) + \frac{\Delta t}{\tau_i} * \left( -y_i + \sum_{j=1}^{N} w_{ji} * \sigma(y_j + \theta_j) + s_i A(x) + g_i B(x;e) \right) \qquad i = 1, 2, \ldots, N \tag{4.1}$$

donde $i$ es un índice ($i = 1, 2, ..., N$), $N$ es el número de neuronas, $y_i$ es el estado de la neurona, $\Delta t$ es el tiempo de ciclo para la integración, $\tau_i$ es la constante temporal de activación de la neurona, $w_{ji}$ es el peso de la conexión entre las neuronas $i$ y $j$, $\theta$ es el término bias, $\sigma(x)$ es la función sigmoidal (ecuación 2.2), $A(x)$ es el sensor de altura, $s_i$ es el peso de de la conexión del sensor de altura, $B(x;e)$ es el sensor de base (el cual depende del tipo de entorno), y $g_i$ es el peso de de la conexión del sensor de base. Al igual que para los experimentos del capítulo anterior, el valor de $\Delta t$ se ha establecido en 0.4 segundos.

Debido a la complejidad del experimento, el proceso de evolución no se realizará desde cero utilizando *Pogamut*, como era el caso de los experimentos del capítulo anterior. Si se pretendiese realizar tal experimento en *Pogamut* para una CTRNN similar, la cual estaría descrita por un genotipo según el cromosoma de 32 genes de la tabla 4.1, dado a que éste no permite paralelizar las evaluaciones de las poblaciones del algoritmo genético (como se ha comentado en la sección 2.1), se estima una duración para el experimento de

$$\begin{aligned} duración \quad = \quad & \frac{200\, ciclos/ejecución * 9\, ejecuciones/individuo * 320\, individuos/generación}{60\, seg/min * 60\, min/hora * 24\, horas/día * 365\, días/año} * \\ & * \frac{320\, generaciones * 0{,}4\, segundos/ciclo}{60\, seg/min * 60\, min/hora * 24\, horas/día * 365\, días/año} = 2,33\, años. \end{aligned} \tag{4.2}$$

Debido a ello, en este proyecto se propone adaptar la CTRNN ya entrenada para el comportamiento de preferencia de temperatura del *C. elegans* al entorno UT2004. Se tomará como punto de partida el modelo de CTRNN que se ha demostrado obtiene mejores resultados para un comportamiento de preferencia de temperatura por parte del *C. elegans*. Dicho modelo consiste en una CTRNN de cuatro neuronas totalmente interconectadas y autoconectadas, las neuronas motoras son simétricamente inversas y con el parámetro temporal de una de las neuronas no motoras más alto que el de las demás (lo que la hace más "lenta"). Se creará una población inicial de 50 individuos, definidos todos ellos por el genotipo que define este modelo CTRNN, a partir de la cual se realizará el proceso de evolución durante 45 generaciones. Se ha modificado ligeramente la técnica de mutación (ver algoritmo 2.1) de manera que

$$u^i_{j,G+1} = x^{r_3}_{j,G} + F(x^{r_1}_{j,G} - x^{r_2}_{j,G}) \qquad \Longrightarrow \qquad u^i_{j,G+1} = x^{r_3}_{j,G} + F1 * x^{r_1}_{j,G} - F2 * x^{r_2}_{j,G}) \ ,$$

ya que si no, al ser idénticos todos los individuos de la población inicial, la mutación tal y como está definida sería un proceso inútil, ya que el término $F(x^{r_1}_{j,G} - x^{r_2}_{j,G})$ siempre devolvería cero. Se estima una duración para el proceso de evolución definido de 5 días y medio.

| **Cromosoma 4 neuronas** | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\tau_1$ | $\theta_1$ | $A_1$ | $B_1$ | $w_{i1}$ | | | | $\tau_2$ | $\theta_2$ | $A_2$ | $B_2$ | $w_{i2}$ | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| $\tau_3$ | $\theta_3$ | $A_3$ | $B_3$ | $w_{i3}$ | | | | $\tau_4$ | $\theta_4$ | $A_4$ | $B_4$ | $w_{i4}$ | | | |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |

Tabla 4.1: Codificación del cromosoma de una CTRNN con neuronas totalmente interconectadas y autoconectadas con 4 neuronas y 2 entradas.

El hecho de tener que realizar el proceso de evolución a partir de una CTRNN obtenida en otro sistema de simulación diferente a UT2004, y por consiguiente con condiciones diferentes (la

CTRNN original fue entrenada para un entorno con un valor $\Delta t$=0.1 y las condiciones de giro y movimiento ligeramente diferentes dado el tamaño del bot), da la oportunidad de comprobar la posibilidad de utilizar para el proceso de evolución entornos de simulación más idóneos que UT2004 utilizando *Pogamut*, que permitan entre otras cosas la paralelización de las evaluaciones y una disminución del tiempo de ciclo para reducir el tiempo de los experimentos.

La función "fitness" utilizada para la evaluación durante el proceso de evolución es

$$\phi = \int_{t=50}^{200} B dt \tag{4.3}$$

donde B es el sensor base. Esta función premia el hecho de que el bot se encuentre en la base enemiga durante las últimas tres cuartas partes de su tiempo de vida.

En cuanto a la configuración del experimento, ésta se encuentra detallada en la tabla 4.2.

| CONFIGURACIÓN EXPERIMENTO | | | |
|---|---|---|---|
| CTRNN | | ALG. GENÉTICO | |
| Neuronas | 2 Entrada | Tamaño Genotipo | 32 |
| | 4 Circuito (2 salida) | Tamaño Población | 50 |
| Simetría | No | Num. Generaciones | 50 |
| $\tau$ | [$e^0$,$e^4$] | Evaluaciones Simultáneas | 4 |
| $\theta$ | [-10,10] | Evaluaciones por bot | 1 |
| $\omega$ | [-10,10] | Tiempo Evaluación | 200 ciclos |
| BOT | | | |
| Ángulo de giro máximo | $45^o$ | | |
| Velocidad Máxima | 0.5 | | |
| Unidad gradiente | 50 Unidades Pogamut | | |

Tabla 4.2: Configuración del experimento para el tercer bot

En la gráfica de la figura 4.3 se pueden ver los valores devueltos por la función "fitness" durante el proceso de evolución. Como se puede ver, mientras que para la CTRNN original el bot obtenía un valor de adaptación máximo del 95 % en UT2004, la mejor CTRNN obtenida tras el proceso de evolución, la cual se muestra en la figura 4.3, alcanza un valor "fitness" del 98,72 %.

## 4.3. Análisis del comportamiento de aprendizaje del bot

Si se analizan los parámetros de la CTRNN obtenida (figura 4.3) se pueden apreciar tres características importantes. En primer lugar, se puede observar una alta simetría bilateral inversa: la forma en la que las neuronas 1 y 2 están conectadas a las neuronas 3 y 4 tienen pesos similares pero de signo contrario; ambas neuronas son de activación rápida; el nodo 3 excita fuertemente a la neurona 1 e inhibe fuertemente a la neurona 2; sus autoconexiones son ambas de pesos pequeños; el sensor de altura inhibe la neurona 1 y excita la 2. Esta simetría es interesante, ya que aunque no se ha impuesto como en el caso de los experimentos del capítulo anterior, ha surgido de la propia evolución dada la estructura del bot. En segundo lugar, el sensor de base tiene una gran fuerza inhibitoria sobre las neuronas motoras, lo cual, como se verá a continuación, permite al bot detenerse una vez ha encontrado la base enemiga. Por último, y quizás la más importante, todas las neuronas actúan lo más rápido posible excepto la neurona 3, la cual es un orden de magnitud menor, lo cual proporciona a la CTRNN una actividad multiescalada en el tiempo.

En la figura 4.4 se muestra el comportamiento del bot en una secuencia con dos cambios de entorno y tres búsquedas de base por entorno, como las utilizadas para la evolución de la CTRNN.

Como se puede observar, al principio de la ejecución el bot se dirige hacia la parte baja del mapa, pero cambia su comportamiento y se dirige hacia arriba antes de alcanzar la región en la

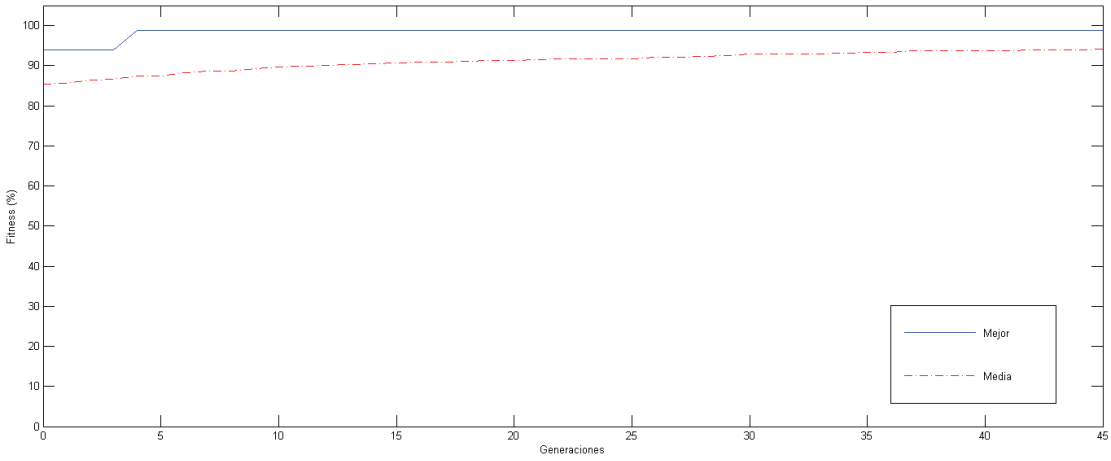
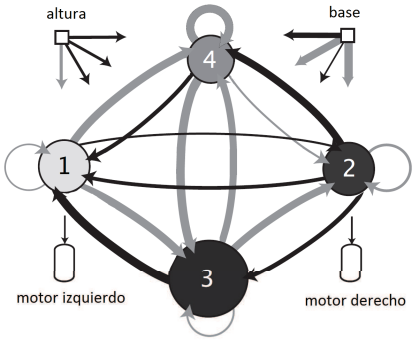



Figura 4.2: Gráfica con los resultados de la función "fitness" obtenidos tras el proceso de evolución para la obtención de una CTRNN con capacidad de aprendizaje en tiempo de ejecución del bot. La línea azul muestra el valor obtenido por el mejor individuo de cada generación. La línea discontinua roja muestra la media de todos los individuos de cada generación.




| | y1 | y2 | y3 | y4 |
|---|---|---|---|---|
| $\tau$ | 1,0521 | 1,023 | 5,8984 | 1,1077 |
| $\theta$ | 5,7900 | -8,415 | -9,3576 | -0,7512 |
| A | -0,621 | 0,6595 | 0,6393 | 0,7439 |
| B | -87358 | -9,5254 | 0,8064 | 4,8860 |
| y1 | -1,8126 | 2,4783 | -9,2366 | -9,1607 |
| y2 | 3,0982 | -3,7070 | 5,3725 | 6,9319 |
| y3 | 9,8654 | -8,6786 | -2,8529 | -8,1644 |
| y4 | 3,4975 | -2,2731 | -9,9652 | -9,4281 |

Figura 4.3: Parámetros para la mejor CTRNN con 4 neuronas totalmente interconectadas y auto-conectadas. Los nodos están sombreados según sus bias. El grosor de las conexiones excitatorias (negras) e inhibitorias (grises) es proporcional al peso de las mismas. Las constantes de tiempo están representados por el tamaño del nodo, siendo las neuronas más lentas las más grandes.

que se encuentra la base de la parte baja del mapa (entre [-10,-9]). Esto parece formar parte de la estrategia de búsqueda, ya que todavía no sabe en qué tipo de entorno se encuentra, y es un fenómeno que se ha observado en todas las ejecuciones del bot. Una vez ha encontrado la base enemiga en la zona alta del mapa, recibe un valor de base B=1 por haber encontrado la base enemiga, y como se puede ver el valor de las neuronas motoras desciende rápidamente a 0 para que el bot permanezca en dicha zona el máximo tiempo posible. Una vez ha encontrado la base enemiga, si se le vuelve a colocar en el centro el bot es capaz de volver a ella de forma más directa.

Una vez el entorno cambia (el bot cambia de equipo), el bot vuelve a la parte alta del mapa, con la diferencia de que recibe un valor del sensor base B=-1 por encontrarse en la base aliada. Como se puede apreciar, el estímulo de B=-1 es muy breve y no parece alterar el valor de los estados de las neuronas, por lo que se puede afirmar que el refuerzo negativo al localizar la base aliada es un estímulo redundante. Lo que el bot hace es seguir subiendo más allá de la base, hasta que cambia gradualmente su comportamiento para dirigirse hacia abajo en el gradiente de la altura, y

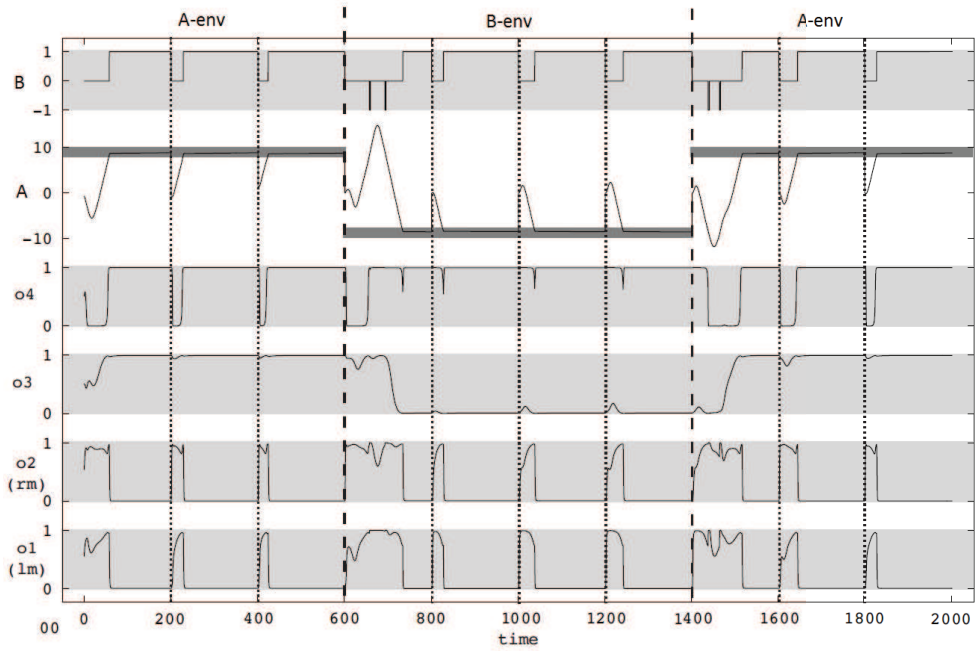



Figura 4.4: Actividad de la CTRNN para una secuencia de ejecución. De arriba a abajo las trazas corresponden a la señal de base (B), la señal de altura (A), y las salidas de las neuronas (oi). Las dos últimas neuronas controlan el motor de la derecha (rm) e izquierda (lm). Las barras horizontales de color gris oscuro en la traza de altura determinan donde puede encontrarse la base enemiga según el entorno A-ent o B-ent. Las líneas discontinuas verticales finas marcan las diferentes ejecuciones (cuando el bot se vuelve a ejecutar desde el centro del mapa). Las líneas discontinuas verticales gruesas marcan la transición entre entornos.

se detiene una vez alcanza la base enemiga, situada esta vez en la parte baja del mapa. Al volver a colocarlo en el centro, el bot ha aprendido dónde se encuentra la nueva base enemiga y de dirige a ella directamente.

Algo similar se observa al volver a cambiar el entorno, lo que demuestra que el agente es capaz de aprender y recordar su comportamiento anterior, así como es capaz de reaprender en caso de que se produzca dicho cambio de entorno.

La explicación a este fenómeno de aprendizaje puede deducirse del comportamiento de las neuronas, concretamente de la neurona 3. Ésta neurona, cuya constante de tiempo es la mayor de todas (figura 4.3), parece ser la única encargada de memorizar el entorno en el que se encuentra el bot. Así pues en caso de encontrarse en un entorno A-ent, la salida o3 se mantendrá en o3=1, pero pasará a ser o3=0 cuando el bot se percate de un cambio al entorno B-ent, y permanecerá en dicho valor hasta que el bot sea consciente de un cambio a A-ent y cambie su valor a o3=1 tal y como se ha explicado previamente.

## 4.4. Análisis de las dinámicas del sistema CTRNN-entorno

Una vez analizados tanto la estructura de la CTRNN como el comportamiento observable del bot al que controla, en esta sección se analizarán las dinámicas del sistema formado por la CTRNN y su entorno. Con ello se pretende entender cómo están estructuradas las dinámicas del bot de forma que el lugar donde la base enemiga fue encontrada en el pasado afecta la dirección

del gradiente de altura en la que se desplazará.

El sistema tiene dos entradas, la altura y el hecho de encontrarse en una de las bases. A pesar de que la altura es una variable contínua, el sensor base es una función discontinua de la altura (ya que su valor depende de la altura a la que se encuentra el bot), lo que hace del agente un sistema dinámico híbrido. Para analizar las dinámicas del sistema, se analizarán los diagramas de bifurcación, los cuales se obtendrán para todos los casos posibles. Así pues, las dinámicas del sistema cambian en función de la altura a la que se encuentra y de si se encuentra o no en la base enemiga (no se tendrá en cuenta el caso en el que el sensor de la base devuelve B=-1, ya que como se ha observado en la sección anterior no afecta al comportamiento del bot).

### Análisis de las dinámicas para B=0

En primer lugar se quiere comprobar cómo cambia el equilibrio del sistema en función de la altura en ausencia de bases. Para ello, se considera un caso sin bases, es decir, un caso en el que el sensor base sea siempre B=0. Dado que el sistema dinámico definido por la CTRNN tiene cuatro variables (la activación de las neuronas) y la altura es el parámetro que cambia, el diagrama de bifurcación es 5-dimensional. En la figura 4.5 se muestran las cuatro proyecciones bidimiensionales del diagrama 5-dimensional, una para cada valor de activación de las neuronas en función de la variable altura.

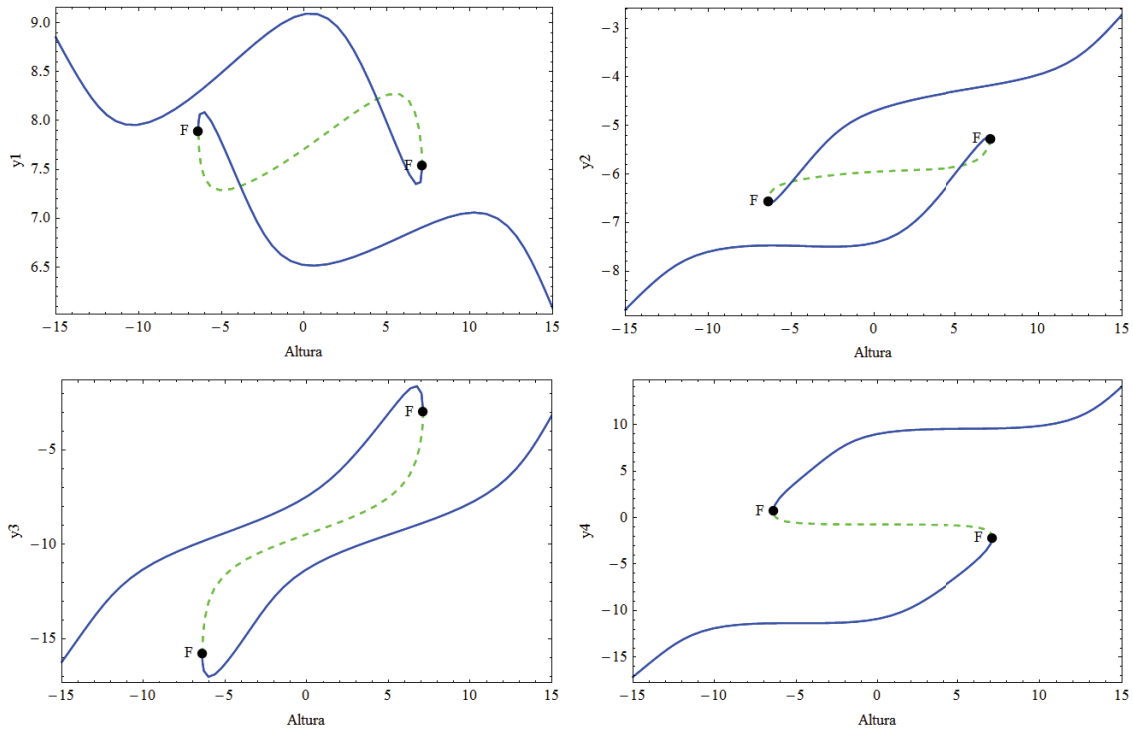



Figura 4.5: Diagrama de bifurcación en ausencia de bases. Cuatro proyecciones bidimensionales del diagrama 5-dimensional, una por cada una de las neuronas de la CTRNN. Las líneas sólidas representan puntos estables de equilibrio, mientras que las líneas discontínuas representan puntos de equilibrio inestables.

### Análisis de las dinámicas para B=1

En segundo lugar, se quiere comprobar como cambia el equilibrio del sistema en función de la altura en presencia de la base enemiga, para lo cual, pese a que se considera una entrada B=1 en todo el rango de alturas, sólo interesan las áreas sombreadas en gris, ya que la base enemiga únicamente puede encontrarse en ellas. En la figura 4.5 se muestran las cuatro proyecciones bidimiensionales del diagrama 5-dimensional para este caso, una para cada valor de activación de las neuronas en función de la variable altura. Como se puede ver, cuando la base enemiga se encuentra en la región correspondiente a alturas "altas", hay un único punto estable de equilibrio, mientras que cuando la base enemiga se encuentra en alturas "bajas", el agente puede estar en uno de dos estados posibles, dado que permanezca allí suficiente tiempo para alcanzar el punto de equilibrio.

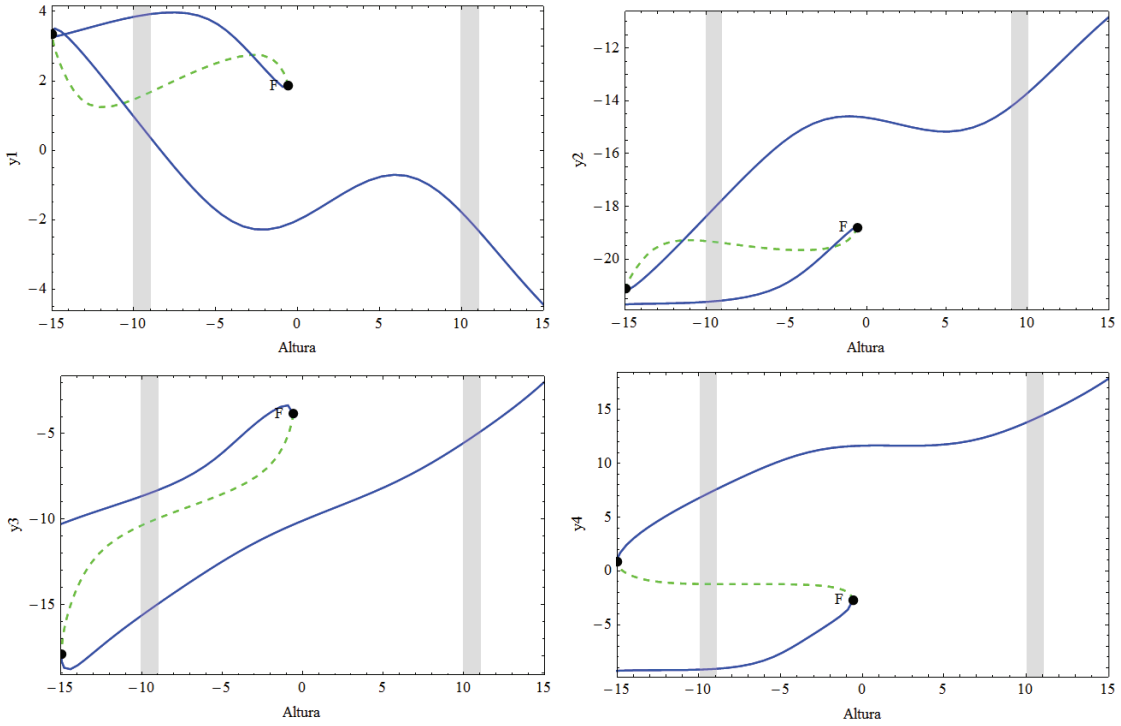



Figura 4.6: Diagrama de bifurcación en presencia de la base enemiga. Cuatro proyecciones bidimensionales del diagrama 5-dimensional, una por cada una de las neuronas de la CTRNN. Las líneas sólidas representan puntos estables de equilibrio, mientras que las líneas discontínuas representan puntos de equilibrio inestables. Las líneas grises verticales muestran los rangos de altura donde puede encontrarse la base enemiga.

### Observaciones

El estudio de éstos diagramas sugiere dos predicciones principales:

1. La primera de ellas es que, en ausencia de cualquier tipo de base, el sistema cae en un ciclo ilimitado en el que el agente alterna entre ir hacia arriba y hacia abajo del gradiente de altura. A pesar de que el bot no ha sido entrenado para este escenario, éste se puede interpretar como un comportamiento de búsqueda "de alto nivel" de la base enemiga que emerge de los comportamientos de "bajo nivel" para los que ha sido entrenada. Este comportamiento

explica el hecho de que no sea necesario considerar el sensor de base cuando éste devuelve un valor -1.

2. En segundo lugar, como resultado de la biestabilidad observada en las gráficas de la figura 4.6, se puede predecir y confirmar que, incluso tras experimentar entornos con la base enemiga en las regiones "bajas", si se expone a alturas bajas en presencia de la base enemiga durante tiempo suficiente, el agente podría llegar a ser recondicionado a navegar hacia arriba en el gradiente de alturas. Esto no sería así en el escenario opuesto, en el que el agente requiere realizar el aprendizaje de la base enemiga en la región baja para recordar. Esto se debe a que el agente está empleando una mezcla de los dos tipos de aprendizaje asociativo:

   *a*) **Condicionamiento clásico:** El agente es capaz de asociar dos estímulos, (altura y base, para aprender dónde se encuentra la base enemiga. Éste tipo de aprendizaje se conoce como condicionamiento clásico (Pavlov, 1927), y se trata del tipo de aprendizaje observado en la sección anterior.

   *b*) **Condicionamiento operativo:** El agente es capaz de asociar un estímulo con un comportamiento, lo que se conoce como condicionamiento operativo (Skinner, 1938). Así pues, el bot asocia el hecho de encontrar la base enemiga, a pesar de encontrarse ésta en la región "baja", con el comportamiento de dirigirse hacia arriba en el gradiente de alturas en busca de dicha base.

## 4.5. Conclusiones: capacidad de memorización en CTRNNs con tiempos de activación multiescala

Tras haber comprobado en el capítulo anterior la capacidad del aprendizaje evolutivo para obtener CTRNNs con capacidades de memoria a corto plazo gracias a las recurrencias entre los nodos de la red, en este capítulo se ha podido comprobar cómo, gracias a la característica de las CTRNNs de que sus neuronas trabajen con diferentes tiempos de activación, un bot es capaz de aprender durante su tiempo de ejecución en el entorno del videojuego UT2004 sin cambiar ninguno de los parámetros de la red. Se ha comprobado además que el comportamiento por parte de la CTRNN para el comportamiento de aprendizaje asociativo para el que se ha entrenado no es siempre tan predecible como a priori pueda parecer. Una vez analizadas las dinámicas del agente con su entorno, se ha visto cómo el agente tenía un comportamiento en el que ignoraba ideas preconcebidas en su diseño: en primer lugar, la CTRNN ignoraba el hecho de estar en la base aliada (indicado por un valor base B=-1) optando por un comportamiento de ascender y descender en el gradiente de alturas en busca de la base enemiga; en segundo lugar, pese a haberse diseñado el experimento según un modelo de aprendizaje asociativo con condicionamiento clásico (asociando altura y base enemiga), la CTRNN muestra la capacidad de aprendizaje asociativo con condicionamiento operativo (asociando estímulo con comportamiento).

En cuanto al entorno de simulación UT2004, se ha comprobado la poca utilidad por parte de *Pogamut* a la hora de evolucionar CTRNNs con muchos parámetros (32 en este caso). No obstante, se ha demostrado que es posible adaptar con éxito al entorno del videojuego UT2004 una CTRNN obtenida en otro entorno de simulación con características diferentes, utilizando para ello la Evolución Diferencial a partir de una población inicial en la que todos los individuos estén definidos por dicha CTRNN. Esto abre la posibilidad de realizar el proceso de evolución en un entorno de simulación con tiempos de ciclo más cortos para la ejecución de las acciones y en el que sea posible paralelizar las evaluaciones de los individuos del algoritmo genético, con lo que se reduciría enormemente el tiempo necesario por parte del algoritmo genético, y una vez obtenida una CTRNN que satisfaga el comportamiento deseado, desplazar el proceso de evolución a *Pogamut* durante unas pocas generaciones para adaptarlo al entorno de UT2004.

# Capítulo 5

# Combinación de CTRNNs para la obtención de un sistema escalable

A la hora de diseñar un bot para un videojuego, no es suficiente con que éste sea capaz de realizar una sola tarea, como es el caso de los bots obtenidos en los capítulos anteriores, los cuales tenían el objetivo de probar características concretas de las CTRNNs como controladores de bots. Por tanto, en este capítulo se buscará obtener bots con capacidad de varios comportamientos. En la sección 5.1 se describe el método utilizado para combinar estos comportamientos y el problema de escalabilidad, que surge a la hora de obtener redes cada vez más complejas. En la sección 5.2 se muestran el diseño de la CTRNN y del experimento de evolución. En la sección 5.3 se muestran los resultados del experimento. Por último, la sección 5.4 recoge las conclusiones extraídas de la ejecución del experimento.

## 5.1. Método utilizado y el problema de la escalabilidad

La escalabilidad y, como consecuencia, la posibilidad de abordar problemas cada vez más complicados, es el gran muro a la hora de obtener agentes autónomos con comportamientos complejos, como afirma Arkin (1998). Los algoritmos genéticos trabajan correctamente cuando los datos son pocos, el espacio de estados es pequeño y es aceptable una búsqueda. Cuando el espacio de estados es inmenso, la utilización de algoritmos genéticos puede llegar a ser inviable (Jakobi, 1998).

Una opción sería la de obtener CTRNNs como controladores de bots que realicen tareas simples, como es el caso de las obtenidas a lo largo de este proyecto, y combinar sus mecanismos para desarrollar ciertas tareas más complejas, pero no es tan evidente. A la hora de combinar dos CTRNNs con comportamientos sencillos, Beer (1995a) explica que no es posible montar las salidas de una sobre la entrada de la otra, ya que esto cambiaría las condiciones iniciales en la ejecución de la red que recibiese la salida. En cuanto al juego de guerra de tanques de Bourquin (2004), éste evoluciona dos CTRNNs, una con comportamiento de navegación en un entorno no estructurado y otra con capacidad para apuntar a tanques enemigos. A la hora de combinar ambos comportamientos, su solución es la de evolucionar la segunda red sobre un tanque con un comportamiento de navegación ya definido. De esta manera, consigue que el segundo comportamiento se adapte al primero, pero no lo contrario.

Debido a que no se han demostrado todavía las pautas a seguir a la hora de obtener CTRNNs complejas como combinación de otras CTRNNs con comportamientos simples, para combinar los comportamientos de dos CTRNNs en un mismo bot, en este capítulo se presenta la siguiente estrategia a seguir: se buscará obtener, mediante aprendizaje evolutivo, una CTRNN capaz de elegir en cada ciclo, según la situación en la que se encuentre, la salida de una de dos CTRNNs, cada una de las cuales posee uno de los comportamientos deseados para el bot.

En concreto, se buscará obtener un bot con los comportamientos proporcionados por las siguientes CTRNNs: (1) la CTRNN con comportamiento de navegación y evitación de obstáculos en un entorno no estructurado, obtenida tras el primer experimento del capítulo 3; (2) la CTRNN obtenida tras el experimento 4, con un comportamiento de búsqueda y memorización de la base enemiga. Se pretende con ello obtener un bot con capacidad de navegación completa, es decir, que sea capaz de buscar y memorizar la posición de la base enemiga durante su ejecución, a la vez que sea capaz de esquivar los obstáculos que encuentre en el entorno.

## 5.2. Diseño del experimento

### Modelado de la CTRNN

Para la obtención del comportamiento de selección deseado para el bot, se ha diseñado una CTRNN como la de la figura 5.1. Como se puede ver lo que se tiene son tres CTRNNs. La de la izquierda tiene un comportamiento de esquivación de los obstáculos que encuentre en el entorno, aprendido en el capítulo 3. La de la derecha tiene un comportamiento de búsqueda y memorización de la posición de la base enemiga, obtenido en el experimento del capítulo 4. Por último, a la CTRNN de en medio, formada por una única neurona autoconectada, se le aplicará la técnica de aprendizaje evolutivo para obtener un comportamiento de selección entre las salidas de las otras dos CTRNNs. Para elo, recibe los valores de algunos los sensores de las dos CTRNNS que componen el sistema: los valores de los sensores raytracing de la CTRNN de la izquierda, cuyos pesos son simétricos, y el valor base de la CTRNN de la derecha. Para un valor de salida o1<0,5, el bot se comporta según las salidas de la CTRNN de la izquierda, y si o1>0,5 se comporta según las salidas de la CTRNN de la derecha.

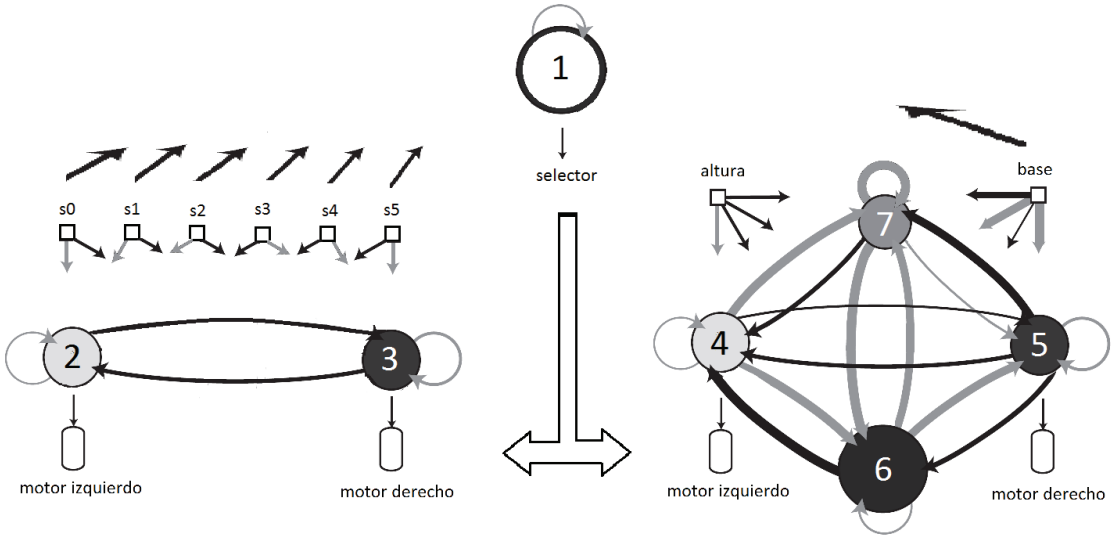



Figura 5.1: CTRNN compuesta por las obtenidas en los experimentos del experimento 1 del capítulo 3 y el del capítulo 4. La neurona 1 está autoconectada, recibe los valores de los sensores base de la CTRNN de la derecha y los valores de los sensores *si* proporcionados por los rayos del sistema "raytracing" de la CTRNN de la izquierda, y se encarga de seleccionar una de las dos CTRNNs para ejecutar las acción del bot.

#### Diseño del mapa para la simulación

El entorno de simulación diseñado corresponde al del mapa de la figura 5.2. Como puede apreciarse, se trata de un entorno similar al del experimento del capítulo 4, pero con obstáculos en el que el bot deberá ser capaz de moverse sin colisionar.

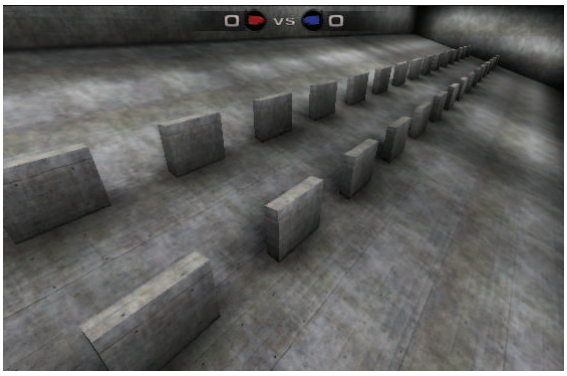


Figura 5.2: Mapas para el experimento de obtención de una CTRNN como combinación de otras CTRNNs.

#### Función "Fitness"

La función "fitness" elegida es

$$\phi = \frac{\sum_{i=1}^{numEjecuciones} (numCiclos - numColisiones_i) * (baseEnemigaEncontrada_i)}{numEjecuciones} \tag{5.1}$$

donde $numEjecuciones$ es el número de búsquedas de la base enemiga, $numCiclos$ es el número de ciclos de vida del bot, $numColisiones_i$ es el número de ciclos que el bot ha permanecido en colisión con un obstáculo, $baseEnemigaEncontrada_i$ tiene un valor de 1 si se ha encontrado la base enemiga en la ejecución $i$ y 0 en caso contrario.

Como se puede apreciar, la función premia el hecho de que el bot sea capaz de encontrar la base enemiga sin colisionar con los obstáculos, lo cual es posible, ya que a pesar de que la CTRNN obtenida en el experimento 1 del capítulo 3 es capaz de regular su velocidad e incluso de detenerse para no colisionar, en caso de que la red elija el otro comportamiento se produciría la colisión. En caso de que el bot no encuentre la base enemiga, el valor sumado al total del valor fitness en esa ejecución es 0, lo cual supone una gran penalización.

#### Configuración de los parámetros para el experimento

La tabla 5.1 recoge los parámetros para la configuración del experimento. Debe tenerse en cuenta que la longitud de los rayos que componen el sistema de sensores "raytracing" es 10 veces el área de colisión del bot (en las unidades de medida utilizadas por *Unreal*).

## 5.3. Resultados del experimento

En la gráfica de la figura 5.3 se pueden ver los valores devueltos por la función "fitness" para el experimento.

El bot que mejor se adapta a la tarea lo hace con un valor "fitness" del 100 %, y sus parámetros se muestran en la tabla 5.2. Para comprobar el correcto funcionamiento de la CTRNN, se ha

| CONFIGURACIÓN EXPERIMENTO 1 | | | | | |
|---|---|---|---|---|---|
| CTRNN | | ALG. GENÉTICO | | RAYTRACING | |
| Neuronas | 7 Entrada | Tamaño Genotipo | 7 | Sensor 0 | -60º |
| | 1 Salida | Tamaño Población | 70 | Sensor 1 | -30º |
| Simetría | Si | Num. Generaciones | 100 | Sensor 2 | -10º |
| $\tau$ | [$e^0$,$e^4$] | Evaluaciones Simultáneas | 4 | Sensor 3 | 10º |
| $\theta$ | [-2,2] | Evaluaciones por bot | 1 | Sensor 4 | 30º |
| $\omega$ | [-5,5] | Tiempo Evaluación | 300 ciclos | Sensor 5 | 60º |

Tabla 5.1: Configuración del experimento para el cuarto bot

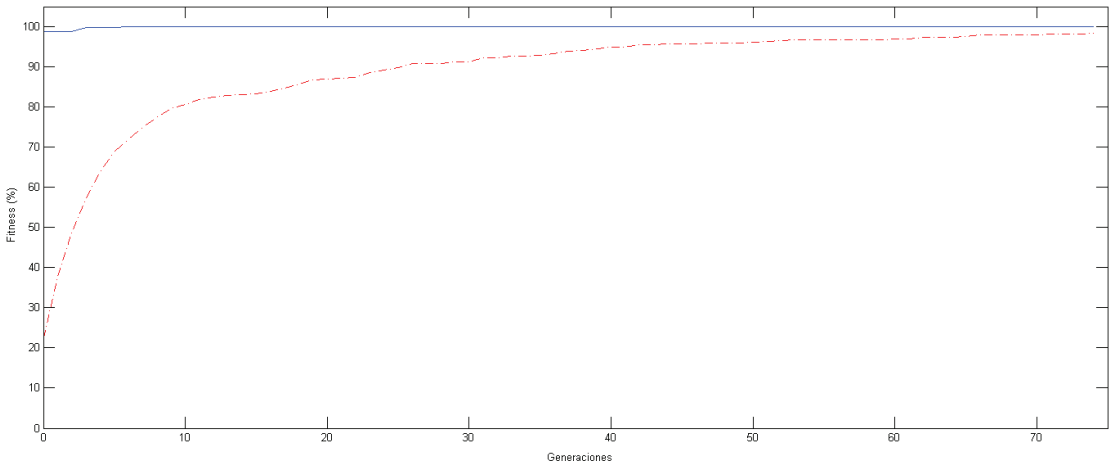



Figura 5.3: Gráfica con los resultados de la función "fitness" obtenidos tras el proceso de evolución para la obtención de una CTRNN capaz de elegir entre el comportamiento de una de las CTRNNs que la componen para con capacidad de navegación y evitación de obstáculos. La línea azul muestra el valor obtenido por el mejor individuo de cada generación. La línea discontinua roja muestra la media de todos los individuos de cada generación.

ejecutado un bot controlado por la misma 100 veces, en las mismas condiciones, con la misma configuración y la misma función "fitness" utilizadas para el aprendizaje evolutivo. Se ha obtenido un éxito del 100 %, por lo que consideramos que la CTRNN se adapta satisfactoriamente a la tarea deseada, ya que el bot controlado por ella ha encontrado la base enemiga en todas sus ejecuciones sin colisionar con ningún obstáculo.

## 5.4. Conclusiones: combinación de comportamientos de CTRNNs

A la hora de obtener bots con varios comportamientos, cada uno de los cuales es realizado por una CTRNN diferente, en este capítulo se ha considerado suficiente el intentar obtener un bot con un comportamiento de selección entre dos comportamientos posibles según la situación en la que se encuentre. Se ha comprobado que es posible obtener una CTRNN capaz de seleccionar en cada ciclo de ejecución entre las salidas de dos CTRNNs que ya sean capaces de realizar los comportamientos deseados. Gracias a ello, se ha conseguido obtener un bot con capacidad de navegación compleja, ya que el bot tiene el objetivo de encontrar la base enemiga en cada una de sus ejecuciones y, además, es capaz de evitar obstáculos en su camino para conseguir dicho objetivo.

| | y1 |
|---|---|
| $\tau$ | 1,0100 |
| $\theta$ | 0,9817 |
| s0 | -3,6816 |
| s1 | -4,6967 |
| s2 | -2,6003 |
| s3 | 1,0035 |
| s4 | 4,6429 |
| s5 | 4,8518 |
| base | -2,1309 |
| y1 | 4,2978 |

Tabla 5.2: Parámetros para la mejor CTRNN que permite selecionar entre los comportamientos de navegación y esquivación de obstáculos por un lado, y de búsqueda y memorización de la localización de la base enemiga por el otro.

Se puede asegurar que ésta capacidad por parte de la CTRNN de selección entre los comportamientos de diferentes CTRNNs es posible para un número de comportamientos reducido. Aunque en este caso se ha probado para dos únicos comportamientos, se presume que ésta capacidad seguiría presente en una CTRNN de dos neuronas totalmente interconectadas y autoconectadas para cuatro comportamientos. No obstante, el uso de la escalabilidad para obtener CTRNNs con comportamientos cada vez más complejos a partir de la combinación de CTRNNs con comportamientos simples todavía es un caso de estudio, por lo que no se puede asegurar que éste sea aplicable según aumenta la complejidad del problema (Jakobi, 1998).

# Capítulo 6

# Conclusiones

## 6.1. Resultados obtenidos

El objetivo de este proyecto era el de obtener bots para el videojuego UT2004 controlados por Redes Neuronales Recurrentes de Tiempo Contínuo (CTRNN), de forma que se estudiasen y aprovechasen las recurrencias entre sus nodos y su actividad multiescalada en el tiempo para obtener bots con comportamientos que serían imposibles si se utilizasen controladores basados en redes neuronales feed-forward. Para ello, se ha utilizado el aprendizaje evolutivo para obtener cuatro bots controlados por CTRNNs:

1. En el capítulo 3 se han obtenido dos bots que utilizan las recurrencias entre los nodos de la CTRNN para obtener comportamientos que requerían memoria a corto plazo:

   *a*) Un primer bot con un comportamiento de navegación y evitación de obstáculos en el entorno no estructurado de UT2004.

   *b*) Un segundo bot con capacidad para seguir la trayectoria de movimiento de un bot enemigo, incluso cuando lo pierde momentáneamente de vista al desaparecer éste tras un muro, para lo cual tendrá que poder "predecir" su reaparición.

2. En el capítulo 4 se ha obtenido un bot controlado por una CTRNN capaz de aprender durante el tiempo de ejecución del bot, sin variar para ello el valor de sus parámetros (CTRNN sin plasticidad sináptica). Para ello, la CTRNN hace uso de la actividad multiescalada en el tiempo para localizar y memorizar la localización de la base enemiga según la altura en que ésta se encuentra.

3. En el capítulo 5 se ha obtenido un bot controlado por una CTRNN capaz de seleccionar entre los comportamientos obtenidos para la CTRNN obtenida en el primer experimento del capítulo 3 y la obtenida en el capítulo 4, según la situación en la que se encuentra. El bot obtenido es capaz de navegar en el entorno con el objetivo de localizar y memorizar la altura a la que se encuentra la base enemiga a la vez que evita los obstáculos que encuentra en su camino.

A continuación, se muestran las conclusiones extraídas tras analizar los resultados de estos experimentos.

## 6.2. Recurrencias entre los nodos de la red para comportamientos con necesidad memoria a corto plazo

Como se ha podido ver en el capítulo 3, las recurrencias entre los nodos de la CTRNN dotan a ésta de una memoria a corto plazo que permite realizar tareas que serían imposibles para una red

neuronal feed-forward. En los experimentos dedicados a la obtención de cada bot se han podido observar las siguientes particularidades:

1. **Primer bot**

   *a*) Es capaz de un comportamiento de navegación y evitación de obstáculos en el entorno no estructurado de UT2004.

   *b*) Se ha comprobado su ventaja respecto a las redes neuronales feed-forward al ser capaz de evitar puntos muertos.

   *c*) A pesar de no haber sido entrenado explícitamente para ello, es capaz de regular su velocidad y evitar puntos muertos sin quedarse bloqueado. Ésto muestra la flexibilidad de este tipo de las CTRNNs para adaptarse a situaciones y comportamientos para los cuales no han sido específicamente entrenadas.

2. **Segundo bot**

   *a*) Las recurrencias entre sus nodos lo dotan de una memoria a corto plazo que le permite seguir su trayectoria de movimiento cuando ha perdido de vista al bot "objetivo" al desaparecer éste tras un muro, lo que le permite "preveer" el momento de su reaparición.

   *b*) Éste tipo de comportamiento sería imposible utilizando redes neuronales feed-forward, ya que al desaparecer el bot "objetivo" tras el muro, el bot controlado por la red no sabría ni en qué dirección ni a qué velocidad girar para reencontrarlo, al depender su comportamiento únicamente de la información recibida en cada ciclo.

Se ha visto como ésta capacidad de memoria a corto plazo, surge incluso cuando se utilizan CTRNNs simétricas tan sencillas como las utilizadas en el capítulo 3, con solo dos neuronas totalmente autoconectadas e interconectadas, y es suficiente para satisfacer los comportamientos de navegación que se buscaban.

## 6.3. Activación multiescalada en el tiempo para un comportamiento de aprendizaje en tiempo de ejecución

En el capítulo 4 se ha podido comprobar lo siguiente:

1. Debido a la característica de las CTRNNs de que sus neuronas trabajen con diferentes tiempos de activación, un bot es capaz de aprender durante su tiempo de ejecución sin cambiar ninguno de los parámetros de la red (CTRNNs sin plasticidad sináptica).

2. Se ha comprobado además que el comportamiento por parte de la CTRNN para realizar satisfactoriamente el comportamiento de aprendizaje asociativo para el que se ha entrenado no es siempre tan predecible como a priori pueda parecer. Una vez analizadas las dinámicas del agente con su entorno, se ha visto cómo el agente tenía un comportamiento en el que ignoraba ideas preconcebidas en su diseño:

   *a*) En primer lugar, la CTRNN resultante tenía un comportamiento según el cual ignoraba el hecho de estar en la base aliada (indicado por un valor base=-1) y optaba por un comportamiento de ascender y descender en el gradiente de alturas en busca de la base enemiga.

   *b*) En segundo lugar, pese a haberse diseñado el experimento según un modelo de aprendizaje asociativo con condicionamiento clásico (asociando altura y base enemiga), la CTRNN obtenida también muestra capacidad de aprendizaje asociativo con condicionamiento operativo (asociando estímulo con comportamiento).

## 6.4. Combinación de CTRNNs para comportamientos complejos

En el capítulo 5, se ha comprobado la posibilidad de obtener un bot con un comportamiento resultante de la combinación de los comportamientos del primer y el tercer bot. Para ello, se ha conseguido obtener una CTRNN con un comportamiento de selección entre uno de estos dos comportamientos según la situación en la que se encuentre.

Del resultado de este experimento, se deduce que es posible obtener una CTRNN capaz de elegir entre las salidas de dos CTRNNs que ya sean capaces de realizar los comportamientos deseados para el bot. Gracias a ello, se ha conseguido obtener un bot con capacidad de navegación compleja, ya que el bot tiene el objetivo de encontrar la base enemiga en cada una de sus ejecuciones y, además, es capaz de evitar obstáculos en su camino para conseguir dicho objetivo.

No obstante, no se puede asegurar que esta técnica de combinación de comportamientos sea aplicable según aumenta la complejidad del problema, ya que el uso de la escalabilidad para obtener CTRNNs con comportamientos cada vez más complejos a partir de la combinación de CTRNNs con comportamientos simples todavía es un caso de estudio (Jakobi, 1998).

## 6.5. Algoritmos Genéticos y CTRNNs en UT2004 utilizando Pogamut

Mediante la realización de este proyecto, se pretendía establecer las bases para abrir un nuevo área de investigación para futuros proyectos, tal y como es el área de la Inteligencia Artificial en videojuegos en general, y la evolución de CTRNNs como controladores de bots en particular:

1. **Programación de IA en videojuegos utilizando Pogamut**

   La primera tarea ha consistido en realizar un estudio de la plataforma *Pogamut* como herramienta de programación de IA para bots en el entorno UT2004. Como primer resultado, desde un punto de vista genérico se ha elaborado el manual del Anexo D, el cual pretende funcionar como manual de consulta a la hora de programar bots en el videojuego UT2004. Las conclusiones obtenidas de la utilización de esta plataforma son las siguiente:

   - *Pogamut* permite programar en Java, nos abstrae de las tareas de conexión al videojuego y nos proporciona herramientas muy útiles para la programación de IA en UT2004 no diponibles en el videojuego (como por ejemplo "raytracing").
   - Aprender a utilizar *Pogamut* supone poder contar con una herramienta pionera y con futuro en el área de investigación de IA en videojuegos, utilizada incluso en el campeonato a nivel mundial *2K BotPrize*.
   - A pesar de sus ventajas, cabe destacar que *Pogamut* todavía está en fase de desarrollo, por lo que todavía contiene errores de programación y produce errores en la ejecución de los bots, lo que debe tenerse en cuenta a la hora de trabajar con él.

2. **Algoritmos genéticos para la obtención de CTRNNs en Pogamut**

   Para la obtención de CTRNNs como controladores de bots utilizando el algoritmo de Evolución Diferencial, del trabajo realizado se deduce lo siguiente:

   - Los experimentos realizados en el capítulo 3 muestran la posibilidad de aplicar la búsqueda de soluciones mediante el aprendizaje evolutivo en el videojuego utilizando *Pogamut* para la obtención de CTRNNs con pocos parámetros.
   - En el experimento realizado en el capítulo 4 se ha comprobado lo siguiente:
     - Se ha comprobado la poca utilidad por parte de *Pogamut* a la hora de evolucionar CTRNNs con muchos parámetros

- Se ha demostrado que es posible adaptar con éxito al entorno del videojuego CTRNNs obtenidas en otros entornos de simulación con características diferentes. Por ello, se recomienda realizar la tarea de evolución utilizando para ello un entorno de simulación con tiempos de ciclo más cortos para la ejecución de las acciones y en el que sea posible paralelizar las evaluaciones de los individuos del algoritmo genético, y una vez obtenida la CTRNN, adaptarla a UT2004 realizando el proceso de evolución durante unas pocas generaciones más utilizando Pogamut.

- Por último, el algoritmo de Evolución Diferencial se ha mostrado como una herramienta óptima para la evolución de las CTRNNs a lo largo del proyecto.

## 6.6. Trabajo futuro

Una vez mostradas las conclusiones extraídas tras la realización de este proyecto, a continuación se muestran algunas de las posibles líneas de trabajo futuro a seguir:

- Una vez estudiadas las capacidades de las CTRNNs para comportamientos que requieren memoria a corto plazo y aprendizaje durante el tiempo de vida del bot, así como la posibilidad de combinar CTRNNs para obtener bots con varios comportamientos, y su utilidad para el control de bots en UT2004, se pueden utilizar dichos conocimientos para la obtención de bots controlados por CTRNNs con comportamientos más complejos.

- Dadas las limitaciones por parte de *Pogamut* para el proceso de aprendizaje evolutivo, se propone la creación de un entorno ligero de simulación, con tiempos de ciclo más cortos para la ejecución de las acciones y en el que sea posible paralelizar las evaluaciones de los individuos del algoritmo genético, orientado a la futura adaptación de las CTRNNs que se obtengan como resultado del proceso evolutivo al entorno del videojuego UT2004.

- Contribuir al desarrollo de la plataforma *Pogamut*, de forma que ésta permita la paralelización de evoluciones de cara al aprendizaje evolutivo.

- Estudiar nuevas propiedades para las CTRNNs o aplicar las estudiadas en este proyecto en entornos de simulación diferentes a los videojuegos.

## 6.7. Valoración personal y problemas encontrados

Al principio de este proyecto, había un total desconocimiento de las técnicas empleadas. Ello requirió una ardua tarea de documentación: (1) para aprender a utilizar la plataforma *Pogamut*; (2) se requirió adquirir grandes conocimientos a cerca de algoritmos genéticos, técnica de la cual tan solo se conocía su existencia, y CTRNNs, para las cuales no existe todavía demasiada información y a penas código proporcionado por otros autores.

En cuanto a la realización del proyecto, cabe destacar la satisfacción de haber utilizado y aprendido acerca de herramientas tan nuevas e innovadoras como Pogamut, haber dado los primeros pasos en la universidad de Zaragoza para la utilización de CTRNNs en el ámbito de los videojuegos, así como el haber adquirido tantos conocimientos acerca de algoritmos genéticos y redes neuronales, herramientas que se tenía un especial interés por aprender.

Los problemas más importantes encontrados a lo largo de la realización del proyecto han sido: (1) el hecho de trabajar con una plataforma en desarrollo como es el caso de *Pogamut*, cuyos problemas de ejecución e impedimento a la hora de paralelizar las evaluaciones alargó considerablemente la vida del proyecto; (2) la falta de documentación por parte de dicha plataforma, la cual en ocasiones era incluso errónea; (3) la falta de ejemplos de código o implementación por parte de los investigadores de CTRNNs, así como el hecho de que el único ejemplo de código que se encontró hacía mal uso de las salidas de la CTRNN, lo que provocó la pérdida de varios meses de ejecución de experimentos.

# Bibliografía


[1] Arkin, R., 1998. Behavior-Based Robotics. MIT Press, Cambridge, MA.

[2] Beer, R. D. (1990). Intelligence as Adaptative Behavior. Academic Press.

[3] Beer, R. D. (1995a). On the dynamics of small continuous-time recurrent neural networks. *Adaptative Behavior*, 3(4) : 459-509.

[4] Beer, R. D. (1995b). A dynamical systems perspective on agent-environment interaction. *Artificial Intelligence*, 72, 173–215.

[5] Beer, R. D. (1996). Towards the Evolution of Dynamical Neural Networks for Minimally Cognitive Behavior. *Adaptive Behavior* 3(4) : 469-509.

[6] Beer, R. D. y Gallagher, J. (1992). Evolving Dynamical Neural Networks for Adaptive Behavior. *Adaptive Behavior*, 1(1), 91–122.

[7] Blynel, J., y Floreano, D. (2002). Levels of dynamics and adaptive behavior in evolutionary neural controllers. En Hallam, B., Floreano, D., Hallam, J., Hayes, G., y Meyer, J.-A. (Eds.), *7th International Conference on Simulation on Adaptive Behavior* (SAB'2002) Edinburgh, UK.

[8] Bourquin, Y. (2004). Tank Wars! Evolving Steering and Aiming Behaviour for Computer Game Agents. Essay in Adaptive Systems, University of Sussex.

[9] Braitenberg V. (1984). *Vehicles*. Cambridge, MA: MIT Press.

[10] Brooks, R. (1990). Elephants don't play chess. *Robotics and Autonomous Systems*, 6, 3–15.

[11] Chapman, N (1999). http://homepages.paradise.net.nz/nickamy/neuralbot/.

[12] Cliff, D., Harvey, I. y Husbands, P. (1993). Explorations in evolutionary robotics. *Adaptive Behavior* 2(1): 73-110.

[13] Collins, R. J. y Jefferson, D. R. (1991). Representations for artificial organisms. En J.-A. Meyer, H. Roitblat and S. Wilson, eds., *From Animals to Animats 1: Proceedings of the Second International Conference on the Simulation of Adaptive Behavior* (pp. 382-390). Cambridge: MIT Press.

[14] de Falco, I., della Cioppa, A., Donnarumma, F., Maisto, D., Prevete, R., Tarantino, E. (2008). A Dierential Evolution Approach to CTRNN Parameter Learning. ECAI 2008: 783-784

[15] de Garis, H. (1992). Steerable GenNets: The genetic programming of steerable behaviors in GenNets. En F.J. Varela and P. Bourgine, eds., *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life* (pp. 272-281), Cambridge: MIT Press.

[16] Floreano, D. y Mondada, F. (1994). Automatic Creation of an Autonomous Agent: Genetic Evolution of a Neural-Network Driven Robot. En D. Cliff, P. Husbands, J. Meyer y S. Wilson, eds., *From Animals to Animats III: SAB'94*. MIT Press-Bradford Books, Cambridge, MA.

[17] Funahashi, K. y Nakamura, Y. (1993). Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks* 6: 801-806.

[18] Gemrot, J., Kadlec, R., Bida, M., Burkert, O., Pibil, R., Havlicek, J., Zemcak, L., Simlovic, J., Vansa, R., Stolba, M., Plch, T., Brom C. (2009). Pogamut 3 Can Assist Developers in Building AI (Not Only) for Their Videogame Agents, In Agents for Games and Simulations, LNCS 5920, Springer. pp. 1-15.

[19] Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading, MA.

[20] Harvey, I., Di Paolo, E., Wood, R., Quinn, M., Tuci, E.A. (2005). Evolutionary robotics: A new scientific tool for studying cognition. *Artificial Life,* 11(1-2) : 79-98.

[21] Holland, J. (1975). Adaptation in Natural and Artificial Systems. U. Michigan Press.

[22] Harvey, I., Husbands, P. y Cliff, D. (1994). Seeing the light: Artificial evolution, real vision. En D. Cliff, P. Husbands y J.-A. Meyer, eds., *From Animals to Animats 3*. Proceedings of the Third International Conference on Simulation of Adaptive Behavior, 392–402. MIT Press, Cambridge, MA.

[23] Hedgecock y Rusell, R. (1975). Normal and mutant thermotaxis in the nematode *Caenorhabditis elegans. Proceedings of the National Academy of Science of the* USA, 72(10), 4061-4065.

[24] Izquierdo, E. (2008). The dynamics of learning behaviour: A situated, embodied, and dynamical systems approach. PhD thesis. COGS, University of Sussex.

[25] Jakobi, N. (1998). Minimal Simulations For Evolutionary Robotics. PhD thesis. COGS, University of Sussex.

[26] Kadlec, R. (2008). Evolution of intelligent agent behaviour in computer games. PhD thesis. Academy of Sciences of the Czech Republic

[27] Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43, 59–69.

[28] Koza, J. R. (1992). Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge, MA.

[29] Laird, J. E., van Lent, M. (2000). Human-level AI's Killer Application: Interactive Computer Games. En Proceedings of AAAI 2000, Austin, USA, pp. 1171-1178.

[30] Langton, C. G. (1989). Artificial Life: the proceedings of an interdisciplinary workshop on the synthesis and simulation of living systems. Addison-Wesley, Redwood City, CA. Workshop held September, 1987 in Los Alamos, New Mexico.

[31] Maass W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural Networks*. 10, 1659-1671.

[32] Matarić M. y Cliff D. (1996). Challenges en Evolving Controllers for Physical Robots. *Evolutional Robotics, special issue of Robotics and Autonomous Systems*, 19(1), 67-83.

[33] Miller, G. F. y Cliff, D. (1994). Protean behavior in dynamic games: Arguments for the coevolution of pursuit-evasion tactics. In D. Cliff, P. Husbands, J. Meyer and S. Wilson, eds., *From Animals to Animats 3: Proceedings of the Second International Conference on the Simulation of Adaptive Behavior* (pp. 411-420). Cambridge: MIT Press.

[34] Nolfi, S y Floreano, D. (2000). Evolutionary Robotics: the biology, intelligence, and technology of self-organizing machines. The MIT Press, Cambridge, MA.

[35] Pavlov, I. (1927). Conditioned Reflexes. Oxford University Press, London.

[36] Price, K.V. (1999). An Introduction to Differential Evolution. En Corne, D., Dorigo, M. y Glover, F. (eds), *New Ideas in Optimization*, McGrawHill, London. pp. 79-108.

[37] Russell, S. J. y Norvig, P. (2003), Artificial Intelligence: A Modern Approach (2nd ed.), Upper Saddle River, New Jersey: Prentice Hall, ISBN 0-13-790395-2.

[38] Skinner, B. F. (1950). Are theories of learning necessary?. Psychological Rewew, 57, 193-216.

[39] Spiessens, P. y Torreele, J. (1992). Massively parallel evolution of recurrent networks: An approach to temporal processing. In F.J. Varela and P. Bourgine, eds., *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life* (pp. 70-77), Cambridge: MIT Press.

[40] Stanley, K. O., Bryant, B. D. y Miikkulainen, R (2005). Evolving Neural Network Agents in the NERO Video Game. En *Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games (CIG'05)*.

[41] Vose, M. D., Wright, A. H. y Row, J. E. (2003). Implicit Parallelism. En E. Cantu-Paz, ed., Proceedings of GECCO 1003–1014. Springer.

[42] Werner, G. M. y Dyer, M. G. (1991). Evolution of communication in artificial organisms. En C. G. Langton, C. Taylor, J. D. Farmer and S. Rasmussen, eds., *Artificial Life II* (pp. 659-687). Reading, MA: Addison-Wesley.

[43] Yamauchi, B. y Beer, R. D. (1994). Sequential behavior and learning in evolved dynamical neural networks. *Adaptive Behavior* 2(3): 219-246.